GOA COLLEGE OF ENGINEERING

"Bhausaheb Bandodkar Technical Education Complex"

Assignment No: 4

JULY 2021 RC 16-17 SOLUTION

SET <u>PART A</u>

Q1)a) State and explain the advantages of distributed systems over centralized system and independent PC's.

Economics

- Microprocessors offer a better price/performance than mainframes
- leading reason for the trend toward distributed systems is that these systems potentially have a much better price/performance ratio than a single large centralized system would have.
- a collection of microprocessors cannot only give a better price/performance ratio than a single mainframe, but may yield an absolute performance that no mainframe can achieve at any price

Speed

- A distributed system may have more total computing power than a mainframe

Inherent distribution

- Some applications involve spatially separated machines
- A supermarket chain might have many stores, each of which gets goods delivered locally (possibly from local farms), makes local sales, and makes local decisions about which vegetables are so old or rotten that they must be thrown out.
- It therefore makes sense to keep track of inventory at each store on a local computer rather than centrally at corporate headquarters.

Reliability

- If one machine crashes, the system as a whole can still survive
- By distributing the workload over many machines, a single chip failure will bring down at most one machine, leaving the rest intact. Ideally, if 5 percent of the machines are down at any moment, the system should be able to continue to work with a 5 percent loss in performance.
- For critical applications, such as control of nuclear reactors or aircraft, using a distributed system to achieve high reliability may be the dominant consideration

Incremental growth

- Computing power can be added in small increments
- Often, a company will buy a mainframe with the intention of doing all its work on it. If the company prospers and the workload grows, at a certain point the mainframe will no longer be adequate. The only solutions are either to replace the mainframe with a larger one (if it exists) or to add a second mainframe. Both of these can wreak major havoc on the company's operations.
- In contrast, with a distributed system, it may be possible simply to add more processors to the system, thus allowing it to expand gradually as the need arises.

b) With a neat diagram explain

1) Bus Based Multiprocessor

- Bus-based multiprocessors consist of some number of CPUs all connected to a common bus, along with a memory module.
- A typical bus has 32 or 64 address lines, 32 or 64 data lines, and perhaps 32 or more control

lines, all of which operate in parallel.

- To read a word of memory, a CPU puts the address of the word it wants on the bus address lines, then puts a signal on the appropriate control lines to indicate that it wants to read.
- The memory responds by putting the value of the word on the data lines to allow the requesting CPU to read it in

2) Switched Multiprocessor

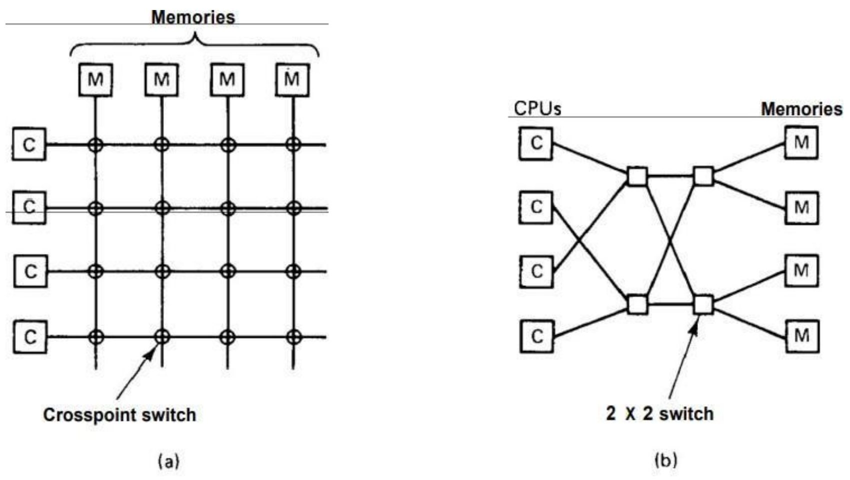


Fig. 1-6. (a) A crossbar switch. (b) An omega switching network.

Crossbar switch

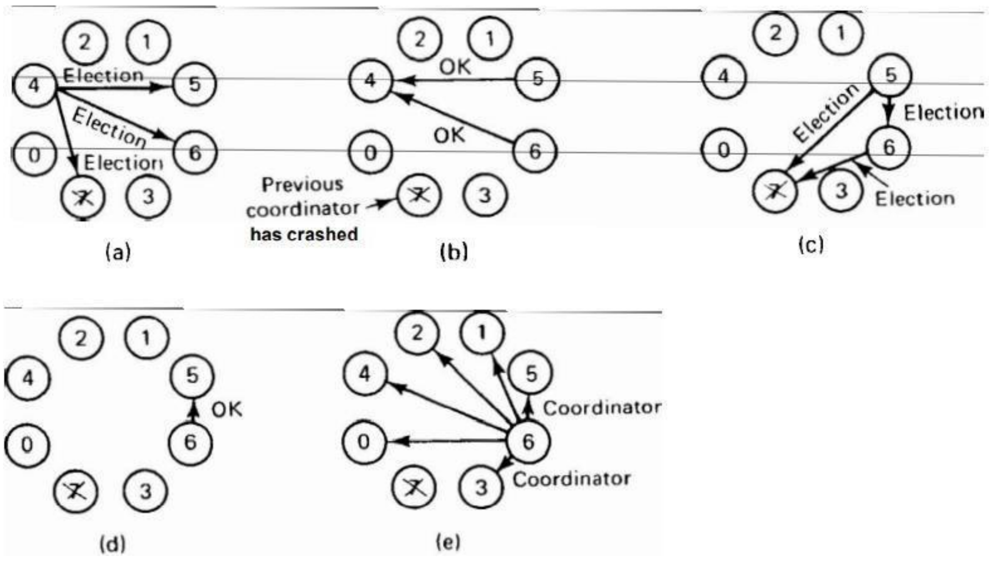
- To build a multiprocessor with more than 64 processors, a different method is needed to connect the CPUs with the memory.
- One possibility is to divide the memory up into modules and connect them to the CPUs with a crossbar switch, as shown in (a)
- At every intersection is a tiny electronic crosspoint switch that can be opened and closed in hardware.
- When a CPU wants to access a particular memory, the crosspoint switch connecting them is closed momentarily, to allow the access to take place.
- The downside of the crossbar switch is that with n CPUs and n memories, n 2 crosspoint switches are needed. For large n, this number can be prohibitive

Omega Network

- Contains four 2 x 2 switches, each having two inputs and two outputs. Each switch can route either input to either output
- In the general case, with n CPUs and n memories, the omega network requires log2n switching stages, each containing nl2 switches, for a total of (n log2n)/2 switches.
- Although for large n this is much better than n2, it is still substantial
- But there is problem of delay.

c) With and example explain the working of Bully algorithm

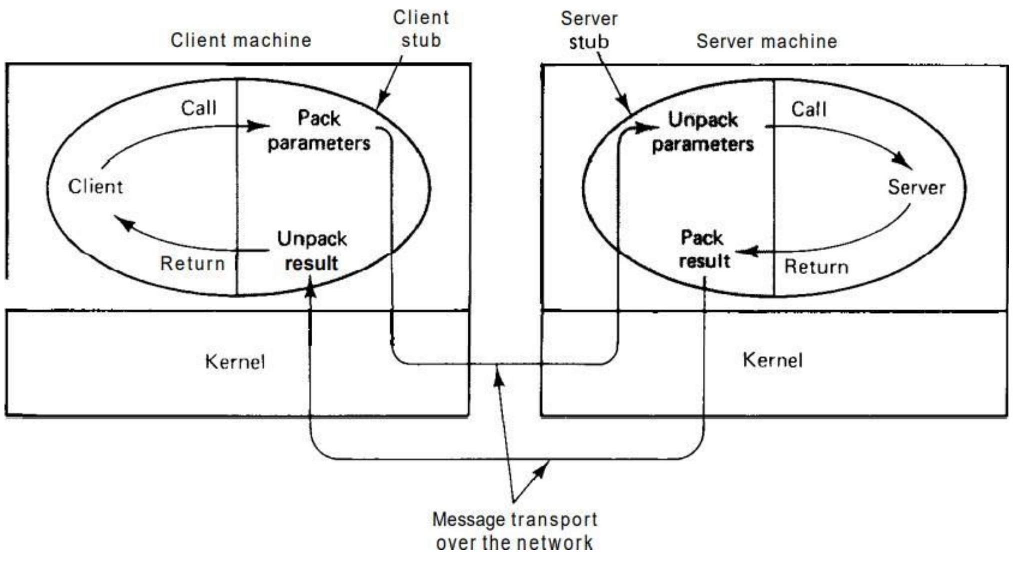
- When a process notices that the coordinator is no longer responding to requests, it initiates an election.
- A process, P, holds an election as follows
  - o P sends an ELECTION message to all processes with higher numbers.
  - o If no one responds, P wins the election and becomes coordinator.
  - o If one of the higher-ups answers, it takes over. P's job is done.



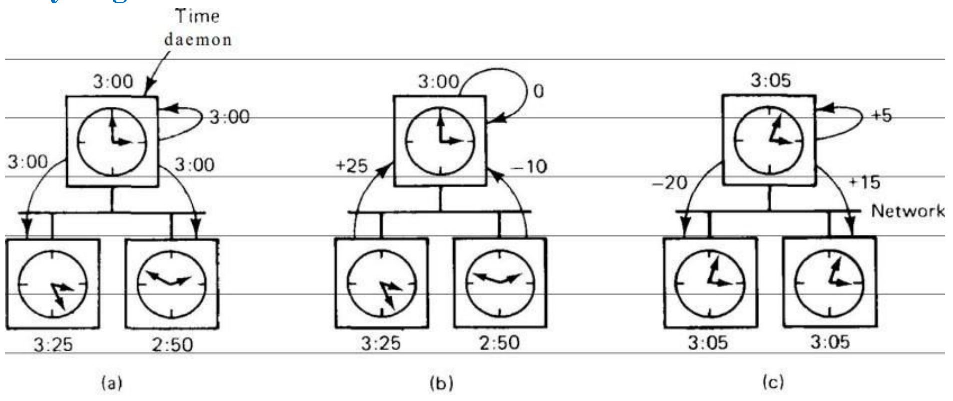
- The group consists of eight processes, numbered from 0 to 7. Previously process 7 was the coordinator, but it has just crashed. Process 4 is the first one to notice this, so it sends ELECTION messages to all the processes higher than it, namely 5, 6, and 7, as shown in (a).
- Processes 5 and 6 both respond with OK, as shown in (b). Upon getting the first of these responses, 4 knows that its job is over. It knows that one of these bigwigs will take over and become coordinator. It just sits back and waits to see who the winner will be
- In (c), both 5 and 6 hold elections, each one only sending messages to those processes higher than itself.
- In (d) process 6 tells 5 that it will take over. At this point 6 knows that 7 is dead and that it is the winner. When it is ready to take over, 6 announces this by sending a COORDINATOR message to all running processes. When 4 gets this message, it can now continue with the operation it was trying to do when it discovered that 7 was dead, but using 6 as the coordinator this time. In this way the failure of 7 is handled and the work can continue
- If process 7 is ever restarted, it will just send all the others a COORDINATOR message and bully them into submission

Q2)a) With a neat diagram explain how a remote procedure call occurs

- The client procedure calls the client stub in the normal way.
- The client stub builds a message and traps to the kernel.
- The kernel sends the message to the remote kernel.
- The remote kernel gives the message to the server stub.
- The server stub unpacks the parameters and calls the server.
- The server does the work and returns the result to the stub.
- The server stub packs it in a message and traps to the kernel.
- The remote kernel sends the message to the client’s kernel.
- The client’s kernel gives the message to the client stub.
- The stub unpacks the result and returns to the client.



b) Explain Berkleys algorithm

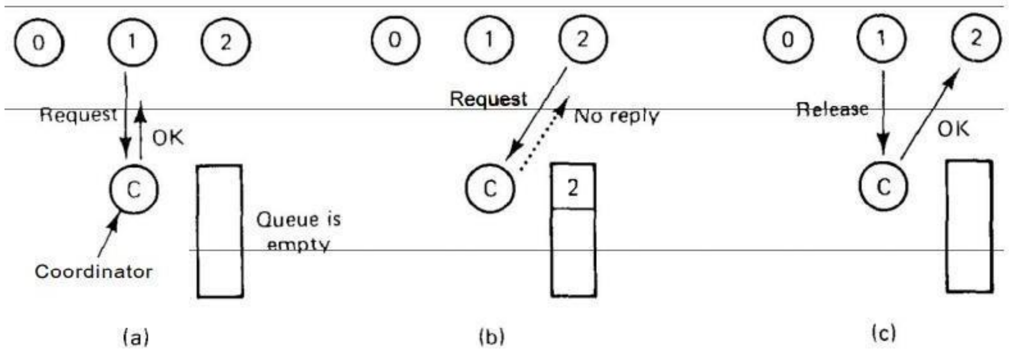


- Here the time server (actually, a time daemon) is active, polling every machine periodically to ask what time it is there.
- Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down until some specified reduction has been achieved.

- This method is suitable for a system in which no machine has a WWV receiver.
- In (a), at 3:00, the time daemon tells the other machines its time and asks for theirs.
- In (b), they respond with how far ahead or behind the time daemon they are.
- Armed with these numbers, the time daemon computes the average and tells each machine how to adjust its clock in (c)

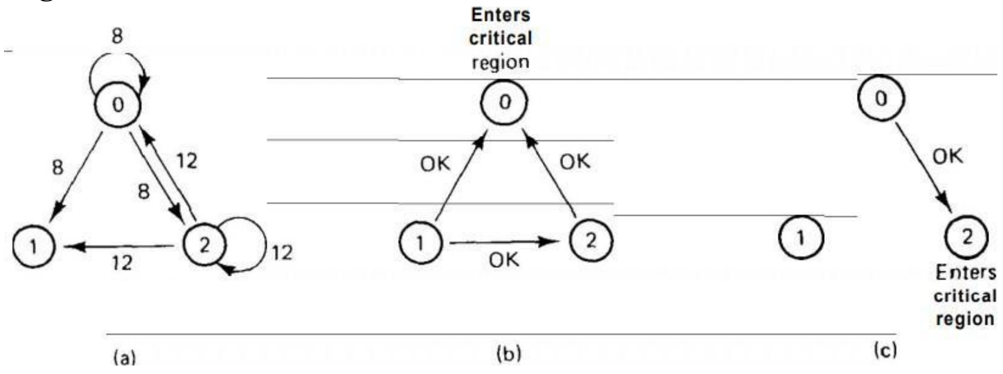
c. Explain any two mutual exclusion algorithm

Centralized Algorithm



- The most straightforward way to achieve mutual exclusion in a distributed system is to simulate how it is done in a one-processor system.
- One process is elected as the coordinator (e.g., the one running on the machine with the highest network address).
- Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission.
- If no other process is currently in that critical region, the coordinator sends back a reply granting permission
- Process 1 asks the coordinator for permission to enter a critical region. Permission is granted.
- Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
- When process 1 exits the critical region, it tells the coordinator, which then replies to 2.

Distributed Algorithm

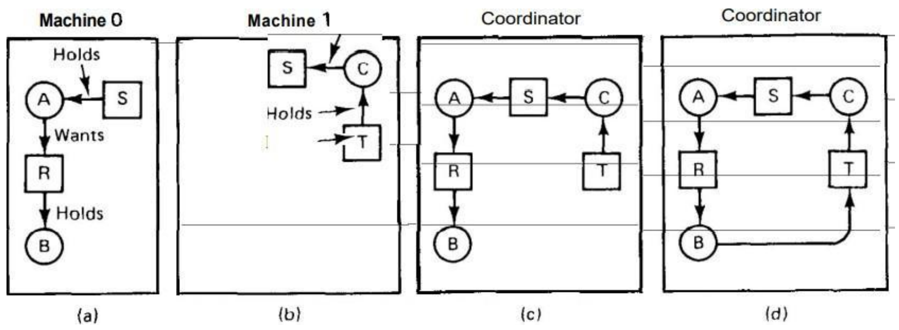


- When a process wants to enter a critical region, it builds a message containing the name of the critical region it wants to enter, its process number, and the current time.

- It then sends the message to all other processes, conceptually including itself. The sending of messages is assumed to be reliable
- When a process receives a request message from another process, the action it takes depends on its state with respect to the critical region named in the message. Three cases have to be distinguished:
    - o If the receiver is not in the critical region and does not want to enter it, it sends back an OK message to the sender.
    - o If the receiver is already in the critical region, it does not reply. Instead, it queues the request.
    - o If the receiver wants to enter the critical region but has not yet done so, it compares the timestamp in the incoming message with the one contained in the message that it has sent everyone. The lowest one wins. If the incoming message is lower, the receiver sends back an OK message. If its own message has a lower timestamp, the receiver queues the incoming request and sends nothing.
- After sending out requests asking permission to enter a critical region, a process sits back and waits until everyone else has given permission. As soon as all the permissions are in, it may enter the critical region. When it exits the critical region, it sends OK messages to all processes on its queue and deletes them all from the queue.
- (a) Two processes 0 and 2 want to enter the same critical region at the same moment.
- (b) Process 0 has the lowest timestamp, so it wins.
- (c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

Q3)a) With an appropriate example explain centralized deadlock detection and distributed deadlock detection algorithms

Centralized Deadlock Detection

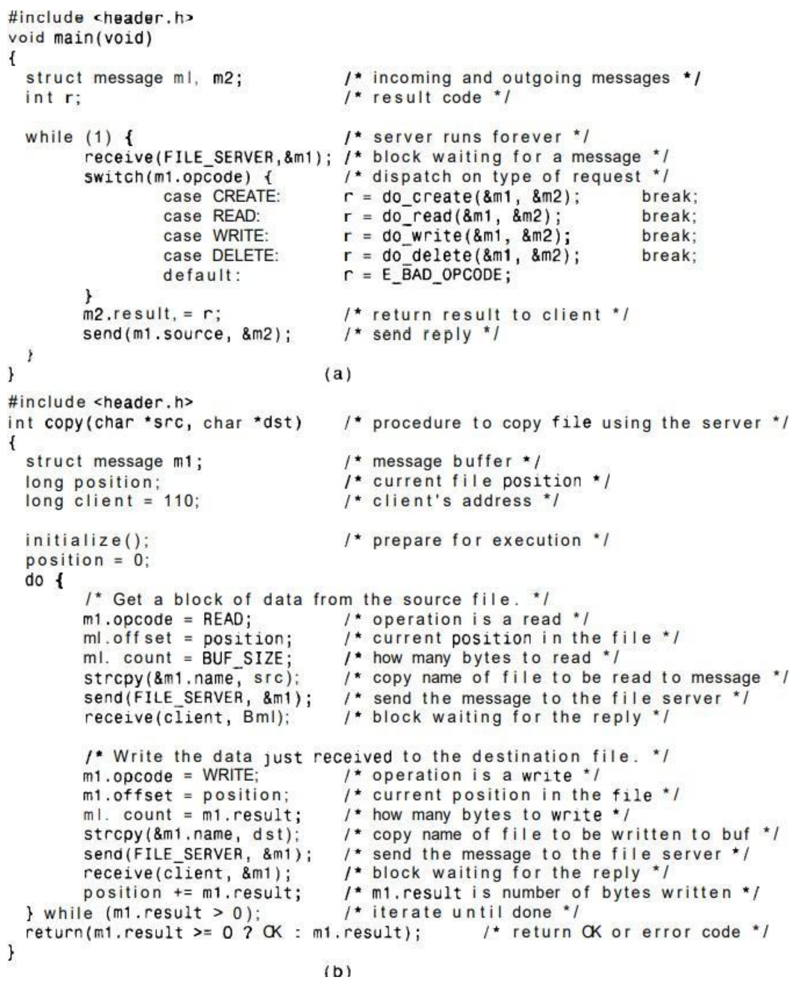


- A central coordinator maintains the resource graph for the entire system
- When the coordinator detects a cycle, it kills off one process to break the deadlock.
- Consider a system with processes A and B running on machine 0, and process C running on machine 1.
- Three resources exist: R, S, and T. Initially, the situation is as show in (a) and (b): A holds S but wants R, which it cannot have because B is using it. C has T and wants S, too
- The coordinator's view of the world is shown in (c). This configuration is safe. As soon as B

finishes, A can get R and finish, releasing S for C.
- After a while, B releases R and asks for T, a perfectly legal and safe swap. Machine 0 sends a message to the coordinator announcing the release of R, and machine 1 sends a message to the coordinator announcing the fact that B is now waiting for its resource, T.
- Unfortunately, the message from machine 1 arrives first, leading the coordinator to construct the graph (d). The coordinator incorrectly concludes that a deadlock exists and kills some process. Such a situation is called a false deadlock.

Distributed deadlock detection
- In Chandy Misra Haas algorithm, processes are allowed to request multiple resources (e.g., locks) at once, instead of one at a time.
- Consider that process 3 on machine 1 is waiting for two resources, one held by process 4 and one held by process 5.
- The Chandy-Misra-Haas algorithm is invoked when a process has to wait for some resource, for example, process 0 blocking on process 1.
- At that point a special probe message is generated and sent to the process (or processes) holding the needed resources
- The message consists of three numbers: the process that just blocked, the process sending the message, and the process to whom it is being sent. The initial message from 0 to 1 contains the triple (0, 0, 1).
- When the message arrives, the recipient checks to see if it itself is waiting for any processes. If so, the message is updated, keeping the first field but replacing the second field by its own process number and the third one by the number of the process it is waiting for.
- If it is blocked on multiple processes, all of them are sent (different) messages.
- If a message goes all the way around and comes back to the original sender, that is, the process listed in the first field, a cycle exists and the system is deadlocked.

b) With the help of a pseudo code explain Client Server Model



```
#include <header.h>
void main(void)
{
  struct message ml, m2;          /* incoming and outgoing messages */
  int r;                          /* result code */

  while (1) {                     /* server runs forever */
        receive(FILE_SERVER,&m1); /* block waiting for a message */
        switch(m1.opcode) {       /* dispatch on type of request */
                case CREATE:      r = do_create(&m1, &m2);      break;
                case READ:        r = do_read(&m1, &m2);        break;
                case WRITE:       r = do_write(&m1, &m2);       break;
                case DELETE:      r = do_delete(&m1, &m2);      break;
                default:          r = E_BAD_OPCODE;
        }
        m2.result, = r;           /* return result to client */
        send(m1.source, &m2);     /* send reply */
  }
}                                 (a)
```

```
#include <header.h>
int copy(char *src, char *dst)    /* procedure to copy file using the server */
{
  struct message m1;              /* message buffer */
  long position;                  /* current file position */
  long client = 110;              /* client's address */

  initialize();                   /* prepare for execution */
  position = 0;
  do {
        /* Get a block of data from the source file. */
        m1.opcode = READ;         /* operation is a read */
        ml.offset = position;     /* current position in the file */
        ml. count = BUF_SIZE;     /* how many bytes to read */
        strcpy(&m1.name, src);    /* copy name of file to be read to message */
        send(FILE_SERVER, &m1);   /* send the message to the file server */
        receive(client, Bml);     /* block waiting for the reply */

        /* Write the data just received to the destination file. */
        m1.opcode = WRITE;        /* operation is a write */
        m1.offset = position;     /* current position in the file */
        ml. count = m1.result;    /* how many bytes to write */
        strcpy(&m1.name, dst);    /* copy name of file to be written to buf */
        send(FILE_SERVER, &m1);   /* send the message to the file server */
        receive(client, &m1);     /* block waiting for the reply */
        position += m1.result;    /* m1.result is number of bytes written */
  } while (m1.result > 0);        /* iterate until done */
  return(m1.result >= 0 ? OK : m1.result);      /* return OK or error code */
}
                                  (b)
```

- Insert the entire contents of header.h into the source program just before the compiler starts compiling the program.
- Header.h starts out by defining two constants, MAXPATH and BUFSIZE, that determine the size of two arrays needed in the message. The former tells how many characters a file may contain.
- The latter fixes the amount of data that may be read or written in one operation by setting the buffer size. The next constant, FILESERVER, provides the network address of the file server so that clients can send messages to it.
- The second group of constants defines the operation numbers. These are needed to ensure that the client and server agree on which code will represent a READ, which code will represent a WRITE, and so on.

Server

- Every reply contains a result code. If the operation succeeds, the result code often contains useful information.If there is no value to be returned, the value OK is used. If the operation is unsuccessful for some reason, the result code tells why, using codes such as EBAD-OPCODE, EXADYARAM, and so on

- The source and dest fields identify the sender and receiver, respectively. The opcode field is one of the operations defined above, that is, CREATE, READ, WRITE, or DELETE.

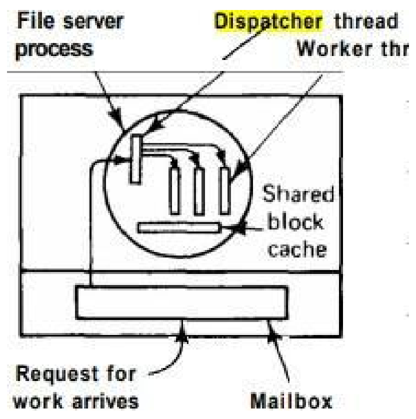
- The server is straightforward. The main loop starts out by calling receive to get a request message.
- The first parameter identifies the caller by giving its address, and the second parameter points to a message buffer where the incoming message can be stored.
- The library procedure *receive* traps to the kernel to suspend the server until a message arrives. When one comes in, the server continues and dispatches on the opcode type.
- For each opcode, a different procedure is called. The incoming message and a buffer for the outgoing message are given as parameters. The procedure examines the incoming message, m1, and builds the reply in m2.
- It also returns a function value that is sent back in the result field. After the send has completed, the server goes back to the top of the loop to execute receive and wait for the next incoming message

Client

- we have a procedure that copies a file using the server. Its body consists of a loop that reads one block from the source file and writes it to the destination file.
- The loop is repeated until the source file has been copied completely, as indicated by a zero or negative return code from the read.
- The first part of the loop is concerned with building a message for the READ operation and sending it to the server.
- After the reply has been received, the second part of the loop is entered, which takes the data just received and sends it back to the server in the form of a WRITE to the destination file

PART B

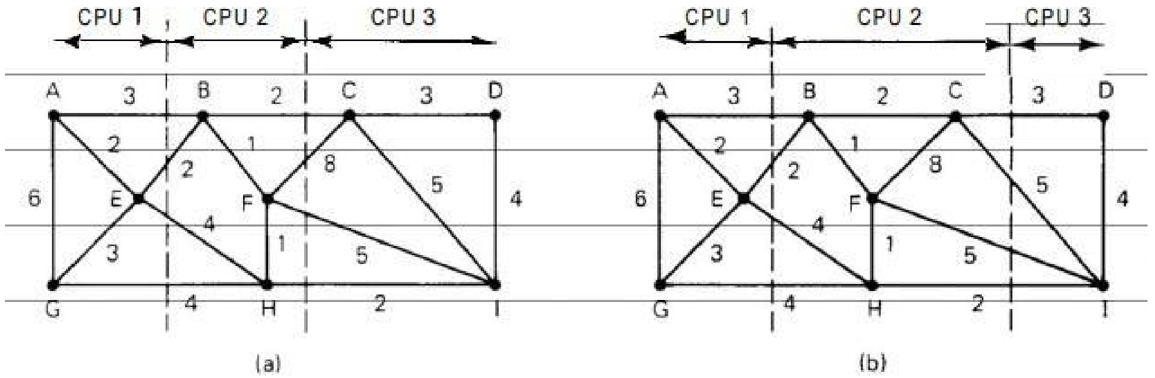
Q4)a)With respect to thread explain the Dispatcher/worker model.



- The dispatcher, reads incoming requests for work from the system mailbox.

- After examining the request, it chooses an idle worker thread and hands it the request.
- The dispatcher then wakes up the sleeping worker
- When the worker wakes up, it checks to see if the request can be satisfied from the shared block cache. If not, it sends a message to the disk to get the needed and goes to sleep awaiting completion of the disk operation.
- The scheduler will now be invoked and another thread will be started, possibly the dispatcher, in order to acquire more work, or possibly another worker that is now ready to run.

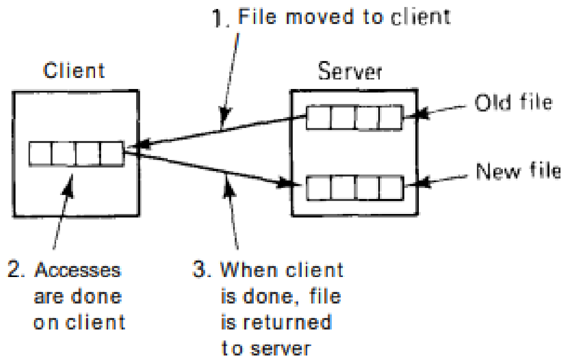
b) With a proper example explain the graph-theoretic deterministic algorithm.



- A widely-studied class of algorithm is for systems consisting of processes with known CPU and memory requirements, and a known matrix giving the average amount of traffic between each pair of processes.
- If the number of CPUs, k, is smaller than the number of processes, several processes will have to be assigned to each CPU. The idea is to perform this assignment such as to minimize network traffic.
- The system can be represented as a weighted graph, with each node being a process and each arc representing the flow of messages between two processes.
- Mathematically, the problem then reduces to finding a way to partition (i.e., cut) the graph into k disjoint subgraphs, subject to certain constraints
- For each solution that meets the constraints, arcs that are entirely within a single subgraph represent intra-machine communication and can be ignored.
- Arcs that go from one subgraph to another represent network traffic.
- The goal is then to find the partitioning that minimizes the network traffic while meeting all the constraints.
- In (a), we have partitioned the graph with processes A, E, and G on one processor, processes B, F, and H on a second, and processes C, D, and I on the third.
- The total network traffic is the sum of the arcs intersected by the dotted cut lines, or 30 units.
- In (b) we have a different partitioning that has only 28 units of network traffic. Assuming that it meets all the memory and CPU constraints, this is a better choice because it requires less communication.

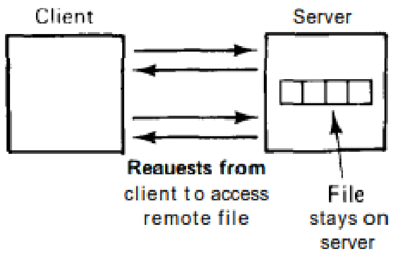
c. Explain the upload/download & the remote access model.

Upload/Download model


1. File moved to client
Client
Server
Old file
New file
2. Accesses are done on client
3. When client is done, file is returned to server

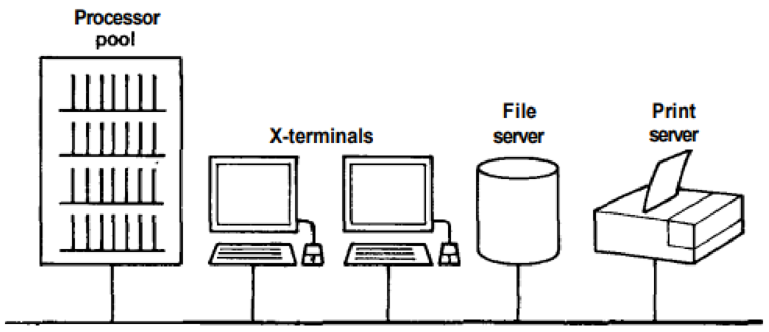
- In the upload/download model, the file service provides only two major operations: read file and write file.
- The former operation transfers an entire file from one of the file servers to the requesting client. The latter operation transfers an entire file the other way, from client to server.
- Thus the conceptual model is moving whole files in either direction. The files can be stored in memory or on a local disk. as needed.
- The advantage of the upload/download model is its conceptual simplicity.
- Application programs fetch the files they need, then use them locally. Any modified files or newly created files are written back when the program finishes.
- No complicated file service interface has to be mastered to use this model. Furthermore, whole file transfer is highly efficient.
- However, enough storage must be available on the client to store all the files required. Furthermore, if only a fraction of a file is needed, moving the entire file is wasteful.

Remote Access model



- In the remote access model , the file service provides a large number of operations for opening and closing files, reading and writing parts of files, moving around within files, examining and changing file attributes, and so on.
- Whereas in the upload/download model, the file service merely provides physical storage and transfer, here the file system runs on the servers, not on the clients.
- It has the advantage of not requiring much space on the clients, as well as eliminating the need to pull in entire files when only small pieces are needed.

Q5)a) Explain the Amoeba system architecture.



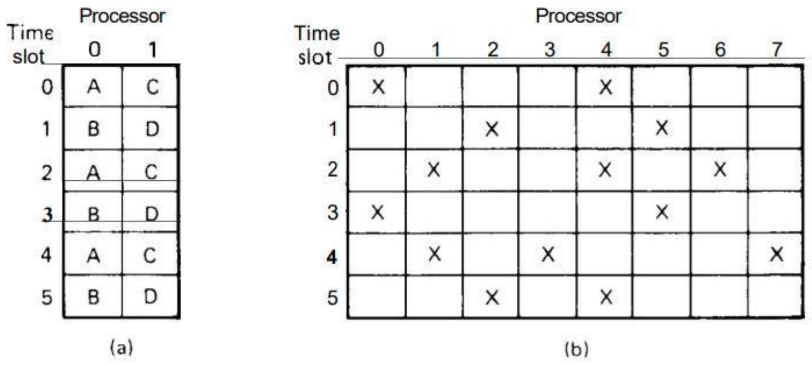
- In this model, all the computing power is located in one or more processor pools.
- A processor pool consists of a substantial number of CPUs, each with its own local memory and network connection.
- Shared memory is not required, or even expected, but if it is present it could be used to optimize message passing by doing memory-to-memory copying instead of sending messages over the network.
- The second element of the Amoeba architecture is the terminal. It is through the terminal that the user accesses the system.
- A typical Amoeba terminal is an X terminal, with a large bit-mapped screen and a mouse.
- Although Amoeba does not forbid running user programs on the terminal, the idea behind this model is to give the users relatively cheap terminals and concentrate the computing cycles into a common pool so that they can be used more efficiently.
- Another important component of the Amoeba configuration consists of specialised servers, such as file servers, which for hardware or software reasons need to run on a separate processor.
- In some cases a server is able to run on a pool processor, being started up as needed, but for performance reasons it is better to have it running all the time.

b) State & explain the various standard operations valid on most objects in Amoeba

- AGE:- The AGE call starts a new garbage collection cycle.
- TOUCH:- The TOUCH call tells the server that the object touched is still in use.
- COPY:- The Copy operation is a shortcut that makes it possible to duplicate an object without actually transferring it. COPY can also fetch remote objects or send objects to remote machines.
- DESTROY:- The DESTROY operation deletes the object. It always needs the appropriate right, for obvious reasons
- GETPARAMS:- Get parameters associated with the server
- SETPARAMS:- Set parameters associated with the server
- INFO:- Get an ASCII string briefly describing the object
- RESTRICT:- Produce a new, restricted capability for the object
- STATUS:- Get current status information from the server

c) Explain how scheduling is handled in distributed computer environment?

Normally, each processor does its own local scheduling (assuming that it has multiple processes running on it), without regard to what the other processors are doing. Usually, this approach works fine. However, when a group of related, heavily interacting processes are all running on different processors, independent scheduling is not always the most efficient way.



Time slot	Processor 0	Processor 1
0	A	C
1	B	D
2	A	C
3	B	D
4	A	C
5	B	D

(a)

Time slot	0	1	2	3	4	5	6	7
0	X				X			
1			X			X		
2		X			X		X	
3	X					X		
4		X		X				X
5			X		X			

(b)

Fig. 4-20. (a) Two jobs running out of phase with each other. (b) Scheduling matrix for eight processors, each with six time slots. The Xs indicated allocated slots.

Difficulty:

Processes A and B run on one processor and processes C and D run on another. Each processor is timeshared in, say, 100-msec time slices, with A and C running in the even slices and B and D running in the odd ones, as shown in Fig. 4-20(a). Suppose that A sends many messages or makes many remote procedure calls to D. During time slice 0, A starts up and immediately calls D, which unfortunate is running because it is now C's turn. After 100 msec, process switching takes place, and D gets A's message, carries out the work, and quickly replies. Because B is now running, it will be another 100 msec before A gets the reply and can proceed. The net result is one message exchange every 200 msec.

Solution: **Use co-scheduling**

It takes interprocess communication patterns into account while scheduling to ensure that all members of a group run at the same time. It uses a conceptual matrix in which each column is the process table for one processor, as shown in Fig. 4-20(b). Thus, column 4 consists of all the processes that run on processor 4. Row 3 is the collection of all processes that are in slot 3 of some processor, starting with the process in slot 3 of processor 0, then the process in slot 3 of processor 1, and so on. The gist of his idea is to have each processor use a round-robin scheduling algorithm with all processors first running the process in slot 0 for a fixed period, then all processors running the process in slot 1 for a fixed period, and so on. A broadcast message could be used to tell each processor when to do process switching, to keep the time slices synchronized.

Q6)a) Briefly explain Amoeba microkernel?

The Amoeba microkernel runs on all machines in the system. The same kernel can be used on the pool processors, the terminals (assuming that they are computers, rather than X terminals), and the specialized servers. The micro- kernel has four primary functions:

1. Manage processes and threads
2. Provide low-level memory management support.
3. Support communication.
4. Handle low-level IO.

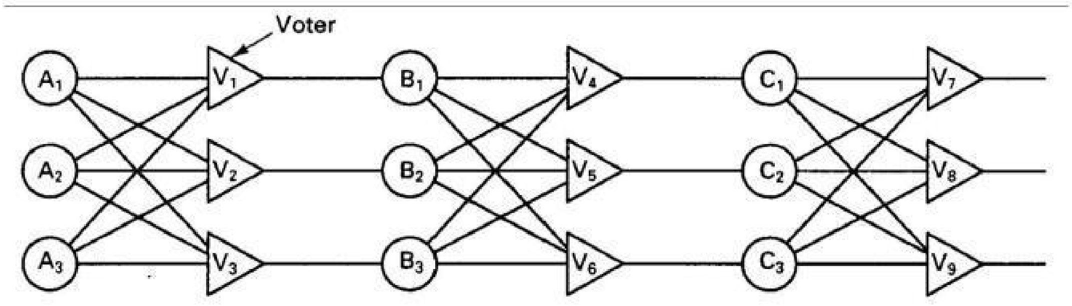
Like most operating systems, Amoeba supports the concept of a process. In addition, Amoeba also supports multiple threads of control within a single address space. A process with one thread is essentially the same as a process in UNIX. Such a process has a single address space, a set of registers, a program counter, and a stack. In contrast, although a process with multiple threads still has a single address space shared by all threads, each thread logically has its own registers, its own program counter, and its own stack.

The second task of the kernel is to provide low-level memory management. Threads can allocate and deallocate blocks of memory, called segments. These segments can be read and written, and can be mapped into and out of the address space of the process to which the calling thread belongs.

The third job of the kernel is to handle interprocess communication. Two forms of communication are provided: point-to-point communication and group communication. These are closely integrated to make them similar.

The fourth function of the kernel is to manage low-level 110. For each IO device attached to a machine, there is a device driver in the kernel. The driver manages all IO for the device. Drivers are linked with the kernel and cannot be loaded dynamically.

b. Explain fault tolerance using active replication and primary backup? **Fault Tolerance Using Active Replication**



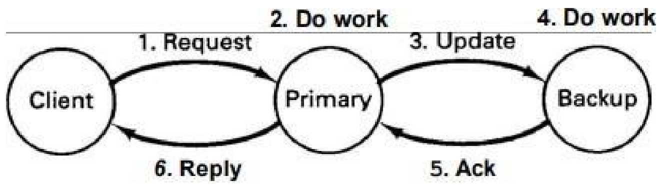
- Active replication is a well-known technique for providing fault tolerance using physical redundancy.

- Some authors refer to active replication as the state machine approach.
- Each device is replicated three times.
- Following each stage in the circuit is a triplicated voter.
- Each voter is a circuit that has three inputs and one output. If two or three of the inputs are the same, the output is equal to that input. If all three inputs are different, the output is undefined.
- This kind of design is known as TMR (Triple Modular Redundancy).

Scenario 1:
Suppose element A fails. Each of the voters, V1 , V2, and V3 gets two good (identical) inputs and one rogue input, and each of them outputs the correct value to the second stage. In essence, the effect of A2 failing is completed masked, so that the inputs to B1, B2, and B3 are exactly the same as they would have been had no fault occurred. Similarly if B3 and C1, are also faulty, in addition to A2, the effects are also masked, so the three final outputs are still correct.

Scenario 2:
Suppose, for example, that V1 malfunctions. The input to B1 will then be wrong, but as long as everything else works, B2 and B3 will produce the same output and V4, V5, and V6 will all produce the correct result into stage three. A fault in V1 is effectively no different than a fault in B1. In both cases B1 produces incorrect output, but in both cases it is voted down later.

Fault Tolerance Using Primary Backup



- The essential idea of the primary-backup method is that at any one instant, one server is the primary and does all the work.
- If the primary fails, the backup takes over.
- Ideally, the cutover should take place in a clean way and be noticed only by the client operating system, not by the application programs.
- The client sends a message to the primary, which does the work and then sends an update message to the backup.
- When the backup gets the message, it does the work and then sends an acknowledgement back to the primary.
- When the acknowledgement arrives, the primary sends the reply to the client.
- It has two major advantages over active replication:
    - o First, it is simpler during normal operation since messages go to just one server (the primary) and not to a whole group.

- o Second, in practice it requires fewer machines, because at any instant one primary and one backup is needed.
- ● On the downside, it works poorly in the presence of Byzantine failures in which the primary erroneously claims to be working perfectly. Also, recovery from a primary failure can be complex and time consuming.

Scenario 1:
If the primary crashes before doing the work (step 2), no harm is done. The client will time out and retry. If it tries often enough, it will eventually get the backup and the work will be done exactly once.

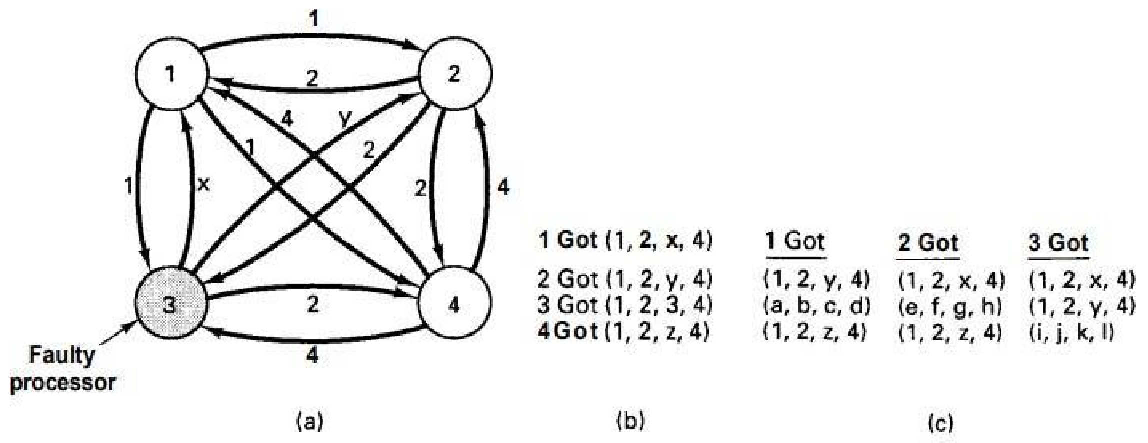
Scenario 2:
If the primary crashes after doing the work but before sending the update, when the backup takes over and the request comes in again, the work will be done a second time.

Scenario 3:
If the primary crashes after step 4 but before step 6, the work may end up being done three times, once by the primary, once by the backup as a result of step 3, and once after the backup becomes the primary.

c) Explain the byzantine general problem with an appropriate example?

There are n generals out of which m of the generals are traitors (faulty) and are actively trying to prevent the loyal generals from reaching agreement by feeding them incorrect and contradictory information (to model malfunctioning processors). The question is now whether the loyal generals can still reach agreement.

We assume each general is assumed to know how many troops he has. The goal of the problem is for the generals to exchange troop strengths, so that at the end of the algorithm, each general has a vector of length n corresponding to all the armies. If general i is loyal, then element i is his troop strength; otherwise, it is undefined.



1 Got (1, 2, x, 4)
2 Got (1, 2, y, 4)
3 Got (1, 2, 3, 4)
4 Got (1, 2, z, 4)

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

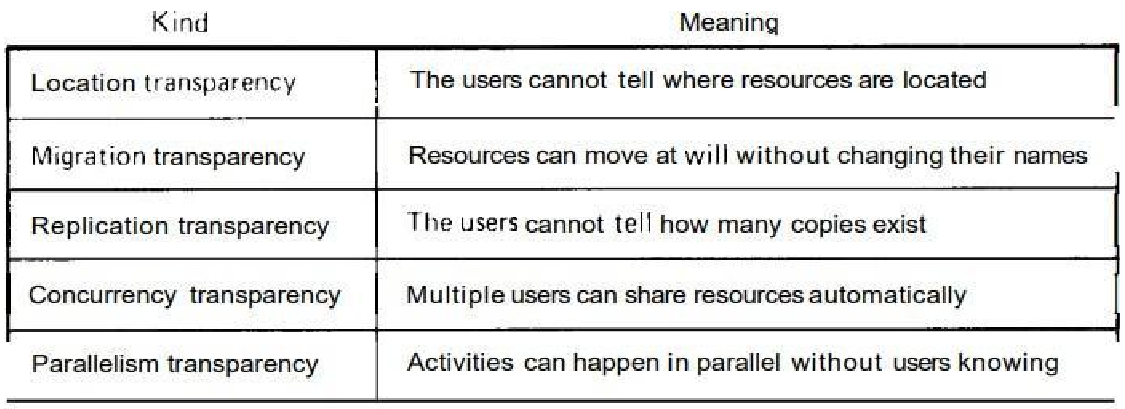
Fig. 4-23. The Byzantine generals problem for 3 loyal generals and 1 traitor. (a) The generals announce their troop strengths (in units of 1K). (b) The vectors that each general assembles based on (a). (c) The vectors that each general receives in step 2.

In Fig. 4-23 we illustrate the working of the algorithm for the case of n = 4 and m = 1. For these parameters, the algorithm operates in four steps.

- In step 1, every general sends a (reliable) message to every other general announcing his truth strength. Loyal generals tell the truth; traitors may tell every other general a different lie. In Fig. 4-23(a) we see that general 1 reports 1K troops, general 2 reports 2K troops, general 3 lies to everyone, giving x, y, and z, respectively, and general 4 reports 4K troops.
- In step 2, the results of the announcements of step 1 are collected together in the form of the vectors of Fig. 4-23(b).
- Step 3 consists of every general passing his vector from Fig. 4-23(b) to every other general. Here, too, general 3 lies through his teeth, inventing 12 new values, a through 1. The results of step 3 are shown in Fig. 4-23(c).
- Finally, in step 4, each general examines the ith element of each of the newly received vectors. If any value has a majority, that value is put into the result vector. If no value has a majority, the corresponding element of the result vector is marked UNKNOWN. From Fig. 4-23(c) we see that general 1, 2, and 4 all come to agreement on
  (1, 2, UNKNOWN, 4)
  which is the correct result. This tells general 3 was the traitor.

<u>PART C</u>

Q7)a) Explain Different Types of transparency in Distributed system



Kind	Meaning
Location transparency	The users cannot tell where resources are located
Migration transparency	Resources can move at will without changing their names
Replication transparency	The users cannot tell how many copies exist
Concurrency transparency	Multiple users can share resources automatically
Parallelism transparency	Activities can happen in parallel without users knowing

Fig. 1-13. Different kinds of transparency in a distributed system.

Location transparency
It refers to the fact that in a true distributed system, users cannot tell where hardware and software resources such as CPUs, printers, files, and data bases are located. The name of the resource must not secretly encode the location of the resource, so names like *machine1:prog.c* or */machine1/prog.c* are not acceptable.

Migration transparency
It means that resources must be free to move from one location to another without having their names change. It allows the user to be unaware of the movement of information or processes within a system without affecting the operations of the users and the applications that are running. This mechanism allows

for the load balancing of any particular client, which might be overloaded. The systems that implement this transparency are NFS.

Replication Transparency
It ensures the existence of numerous instances of resources to improve reliability and performance without having to know about replication to the user. In other words, this type of transparency should primarily be applied to distributed file systems, where replication of data over two or more sites exists for increased reliability. The existence of a mirrored copy of data must be unknown to the client. Example include DDBMS.

Concurrency Transparency
Users and Applications should be able to access shared data or objects without interference between each other. This requires very complex mechanisms in a distributed system, since there exists true concurrency rather than the simulated concurrency of a central system. The shared objects are accessed simultaneously. The concurrency control and its implementation is a hard task. The examples are NFS, Automatic Teller machine (ATM) network.

Parallelism Transparency
Parallelism Transparency enables parallel activities to run without users knowing how, where and when it is made possible by the systems. A distributed system must have automatic use of parallelism without having to program it explicitly.

b) Explain different types of addressing in a client server model.

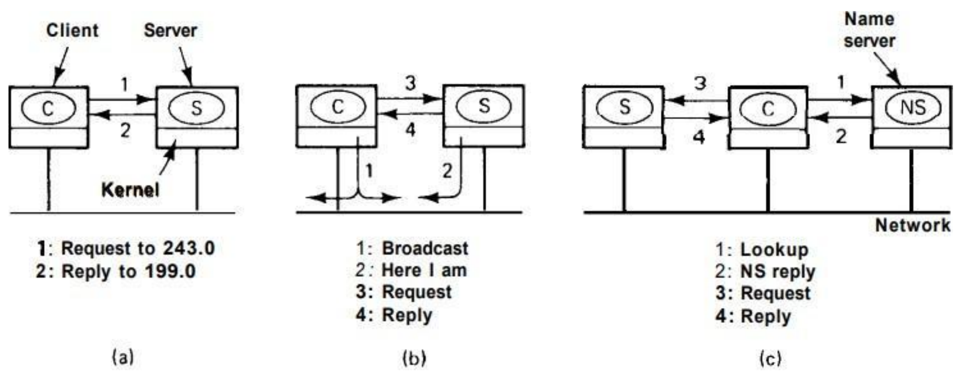


Fig. 2-10. (a) Machine.process addressing. (b) Process addressing with broadcasting. (c) Address lookup via a name server.

Strategy 1 - Hardwire machine.numher into client code.

The addressing system sends messages to processes rather than to machines. Although this method eliminates all ambiguity about who the real recipient is, it introduces the problem of how processes are identified.

One common scheme is to use two part names, specifying both a machine and a process number. Thus

243.4 or 4@243 or something similar designates process 4 on machine 243. The machine number is used by the kernel to get the message correctly delivered to the proper machine, and the process number is used by the kernel on that machine to determine which process the message is intended for. A nice feature of this approach is that every machine can number its processes starting at 0. No global coordination is needed because there is never any ambiguity between process 0 on machine 243 and process 0 on machine 199. The former is 243.0 and the latter is 199.0 illustrated in fig 2.10 (a)

Disadvantage: It is not transparent to the user. If the server is changed from 243 to 170, the program must be changed.

Strategy 2 – Let processes pick random addresses; locate them by broadcasting.

Each each process picks its own id from a large, sparse address space, such as the space of 64-bit binary integers. The probability of two processes picking the same number is tiny, and the system scales well. However, here, too, there is a problem: How does the sending kernel know what machine to send the message to?

On a LAN that supports broadcasting, the sender can broadcast a special locate packet containing the address of the destination process. Because it is a broadcast packet, it will be received by all machines on the network. All the kernels check to see if the address is theirs, and if so, send back a here I am message giving their network address (machine number). The sending kernel then uses this address, and furthermore, caches it, to avoid broadcasting the next time the server is needed. This method is shown in Fig. 2-10 (b)

Disadvantage: Broadcasting puts extra load on the system.

Strategy 3 - Put ASCII server names in clients; look them up at run time.

The broadcasting load can be avoided by providing an extra machine to map high-level (ASCII) service names to machine addresses. As shown in Fig. 2-10(c), when this system is employed, processes such as servers are referred to by ASCII strings, and it is these strings that are embedded in programs, not binary machine or process numbers. Every time a client runs, on the first attempt to use a server, the client sends a query message to a special mapping server, often called a name server, asking it for the machine number where the server is currently located. Once this address has been obtained, the request can be sent directly. As in the previous case, addresses can be cached.

Disadvantage: The centralized component - name server

c) Explain the two-phase commit protocol.

Two phase commit protocol is used for achieving atomic commit in a distributed system. The basic idea is illustrated in Fig. 3-20

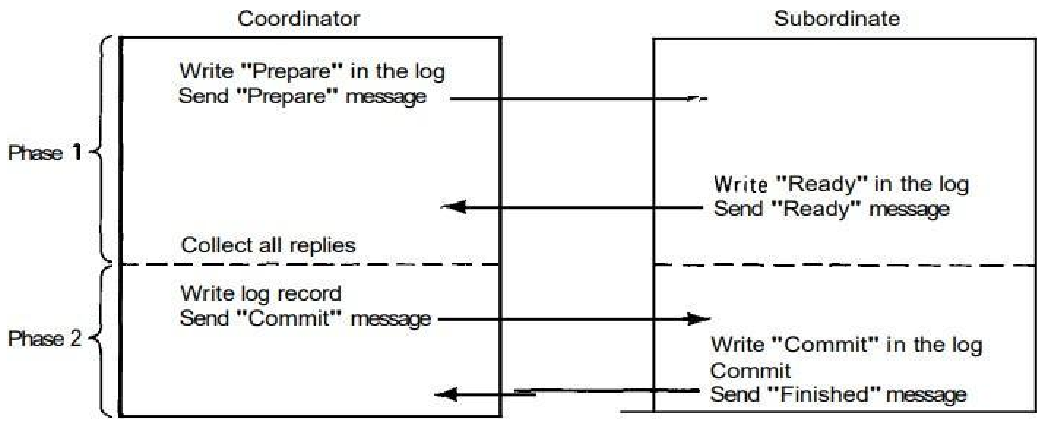


Fig. 3-20. The two-phase commit protocol when it succeeds.

One of the processes involved functions as the coordinator. Usually, this is the one executing the transaction. The commit protocol begins when the coordinator writes a log entry saying that it is starting the commit protocol, followed by sending each of the other processes involved (the subordinates) a message telling them to prepare to commit.

When a subordinate gets the message, it checks to see if it is ready to commit, makes a log entry, and sends back its decision. When the coordinator has received all the responses, it knows whether to commit or abort. If all the processes are prepared to commit, the transaction is committed. If one or more are unable to commit (or do not respond), the transaction is aborted. Either way, the coordinator writes a log entry and then sends a message to each subordinate informing it of the decision. It is this write to the log that actually commits the transaction and makes it go forward no matter what happens afterward.

Advantages:

Due to the use of the log on stable storage, this protocol is highly resilient in the face of (multiple) crashes. If the coordinator crashes after having written the initial log record, upon recovery it can just continue where it left off, repeating the initial message if need be. If it crashes after having written the result of the vote to the log, upon recovery it can just reinform all the subordinates of the result. If a subordinate crashes before having replied to the first message, the coordinator will keep sending it messages, until it gives up. If it crashes later, it can see from the log where it was, and thus what it must do.

Q8)a) With a neat diagram explain the process descriptor in Amoeba

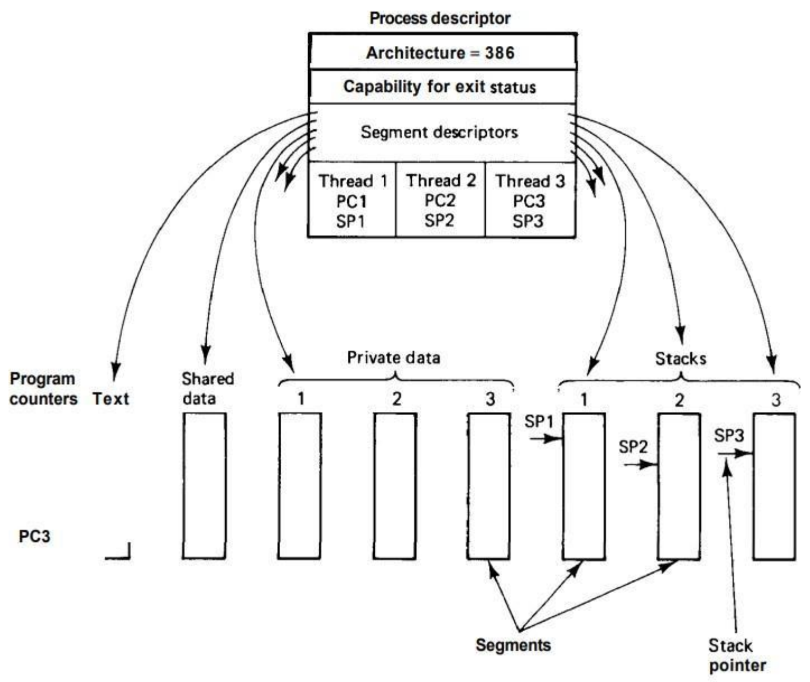


Fig. 7-6. A process descriptor.

A process descriptor is a data structure that provides information about the process to be run. One field in the process descriptor (see Fig. 7-6) tells which CPU architecture the process can run on. In heterogeneous systems, this field is essential to make sure that 386 binaries are not run on SPARCs, and so on.

Another field contains the process' owner's capability. When the process terminates or is stunned (see below), RPCs will be done using this capability to report the event. It also contains descriptors for all the process' segments, which collectively define its address space, as well as descriptors for all its threads.

Finally, the process descriptor also contains a descriptor for each thread in the process. The content of a thread descriptor is architecture dependent, but as a bare minimum, it contains the thread's program counter and stack pointer. It may also contain additional information necessary to run the thread, including other registers, the thread's state, and various flags. Brand new processes contain only one thread in their process descriptors, but stunned processes may have created additional threads before being stunned.

b) Explain any two methods for reliable broadcasting in Amoeba

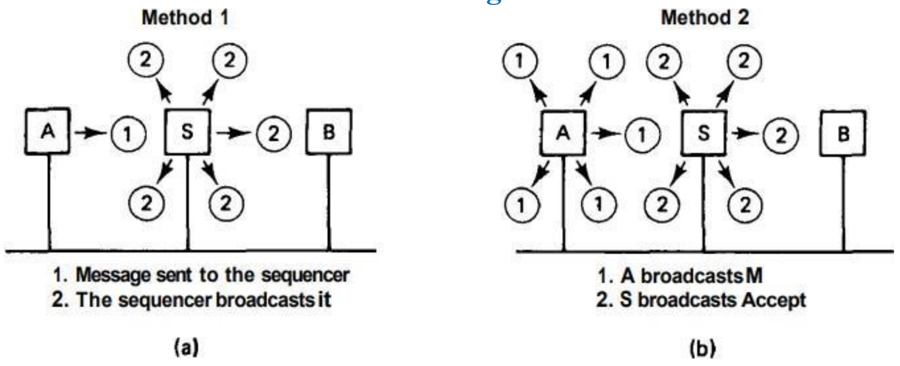


Fig. 7-13. Two methods for doing reliable broadcasting.

Method 1: Fig 7-13 (a)

The users sends a point-to-point Request for Broadcast message to the sequencer which carries a piggybacked acknowledgement, k, meaning that all broadcasts up to and including k have been correctly received. This way, the sequencer can maintain a piggyback table, indexed by machine number, telling for each machine which broadcast was the last one received.

If one machine happens to be silent for an unusually long period of time, the sequencer will not know what its status is. To inform the sequencer, it is required to send a short acknowledgement message when it has sent no broadcast messages for a certain period of time. Furthermore, the sequencer can broadcast a *Request for Status* message, which directs all other machines to send it a message giving the number of the highest broadcast received in sequence.

Performance:

In this method, each message appears in full on the network twice: once to the sequencer and once from the sequencer. Thus a message of length *m* bytes consumes *2m* bytes worth of network bandwidth. However, only the second of these is broadcast, so each user machine is interrupted only once (for the second message).

Method 2: Fig 7-13 (b)

This method is logically equivalent to the first method with some modifications. In this method, the user broadcasts the message, including a unique identifier. When the sequencer sees this, it broadcasts a special Accept message containing the unique identifier and its newly assigned sequence number. A broadcast is "official" only when the Accept message has been sent.

Performance:

In this method, the full message appears only once on the network, plus a very short *Accept* message from the sequencer, so only half the bandwidth is consumed. On the other hand, every machine is interrupted twice, once for the message and once for the *Accept.* Thus method 1 wastes bandwidth to reduce interrupts compared to method 2. Depending on the average message size, one may be preferable to the other.

c) Explain in brief Directory Server Interface

- All distributed systems allow directories to contain subdirectories, to make it possible for users to group related files together.
- Accordingly, operations are provided for creating and deleting directories as well as entering, removing, and looking up files in them.
- Subdirectories can contain their own subdirectories, and so on, leading to a tree of directories, often called a hierarchical file system. Figure 5-2(a) illustrates a tree with five directories.
- In some systems, it is possible to create links or pointers to an arbitrary directory. These can be put in any directory, making it possible to build not only trees, but arbitrary directory graphs, which are more powerful. (Fig 5-2(b))
- In a tree-structured hierarchy, a link to a directory can be removed only when the directory pointed to is empty.
- In a graph, it is allowed to remove a link to a directory as long as at least one other link exists.

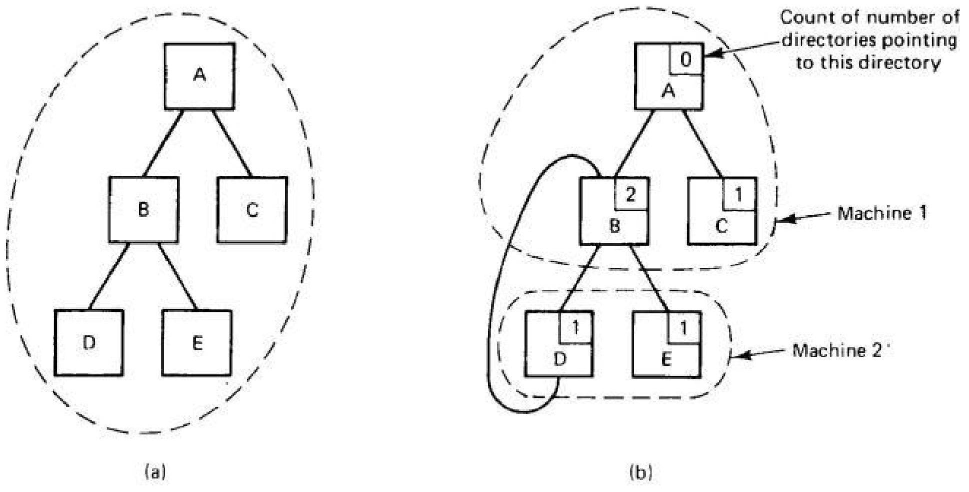
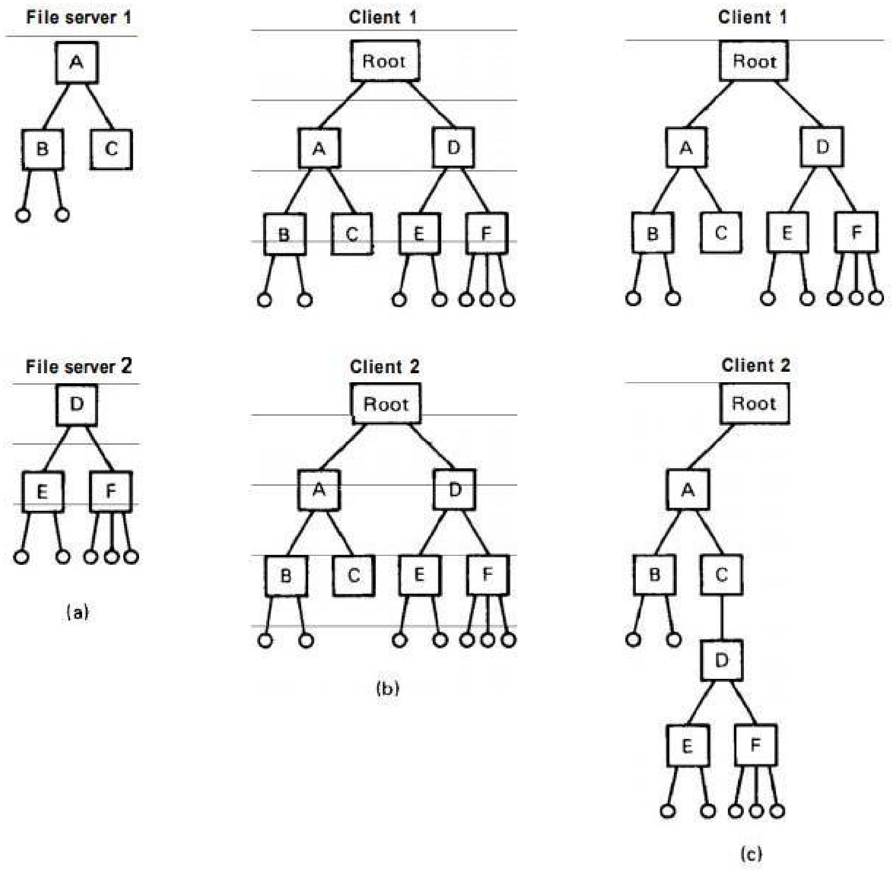
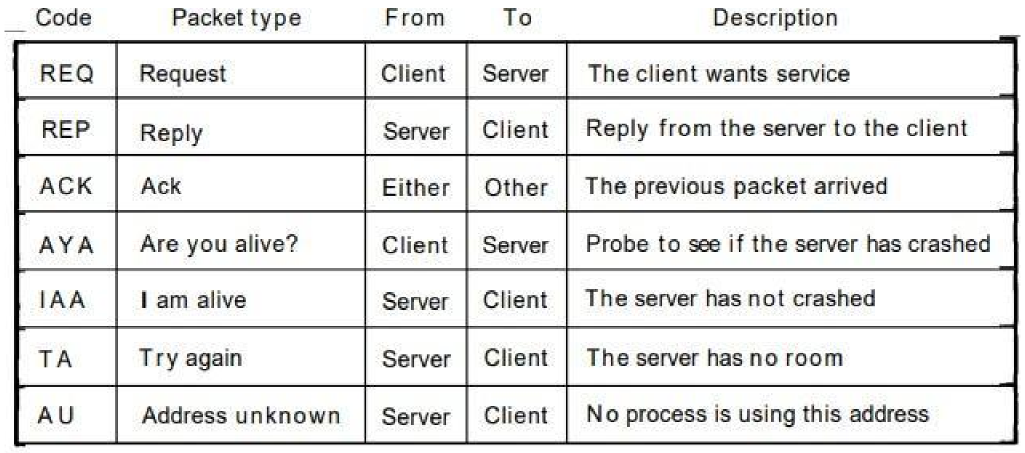


Fig. 5-2. (a) **A** directory tree contained on one machine. (b) **A** directory graph on two machines.

- A key issue in the design of any distributed file system is whether or not all machines (and processes) should have exactly the same view of the directory hierarchy. An example is shown in fig below
  Fig (a) Two file servers. The squares are directories and the circles are files.
  Fig (b) A system in which all clients have the same view of the file system.
  Fig (c) A system in which different clients may have different views of the file system.



d) State and explain the different packet types used in the client server protocol.



Code	Packet type	From	To	Description
REQ	Request	Client	Server	The client wants service
REP	Reply	Server	Client	Reply from the server to the client
ACK	Ack	Either	Other	The previous packet arrived
AYA	Are you alive?	Client	Server	Probe to see if the server has crashed
IAA	I am alive	Server	Client	The server has not crashed
TA	Try again	Server	Client	The server has no room
AU	Address unknown	Server	Client	No process is using this address

Fig. 2-15. Packet types used in client-server protocols.

REQ, REP, ACK
REQ packet is used to send a request message from a client to a server. The next one is the REP packet that carries results back from the server to the client. Then comes the ACK packet, which is used in

reliable protocols to confirm the correct receipt of a previous packet.

AYA, IAA

Consider the situation in which a request has been sent successfully from the client to the server and the acknowledgement has been received. At this point the client's kernel knows that the server is working on the request. But what happens if no answer is forthcoming within a reasonable time? Is therequest really that complicated, or has the server crashed? To be able to distinguish these two cases, the AYA packet is sometimes provided, so the client can ask the server what is going on. If the answer is IAA, the client's kernel knows that all is well and just continues to wait. The AYA and IAA packets can also be used even in a protocol in which REQ packets are not acknowledged.

TA, AU

The last two packet types, which are useful in case a REQ packet cannot be accepted. There are two reasons why this might happen, and it is important for the client's kernel to be able to distinguish them. One reason is that the mailbox to which the request is addressed is full. By sending this packet back to the client's kernel, the server's kernel can indicate that the address is valid, and the request should be repeated later. The other reason is that the address does not 'belong to any process or mailbox. Repeating it later will not help.