1.Explain with example, how wait-for-graphs can be used in deadlock detection.

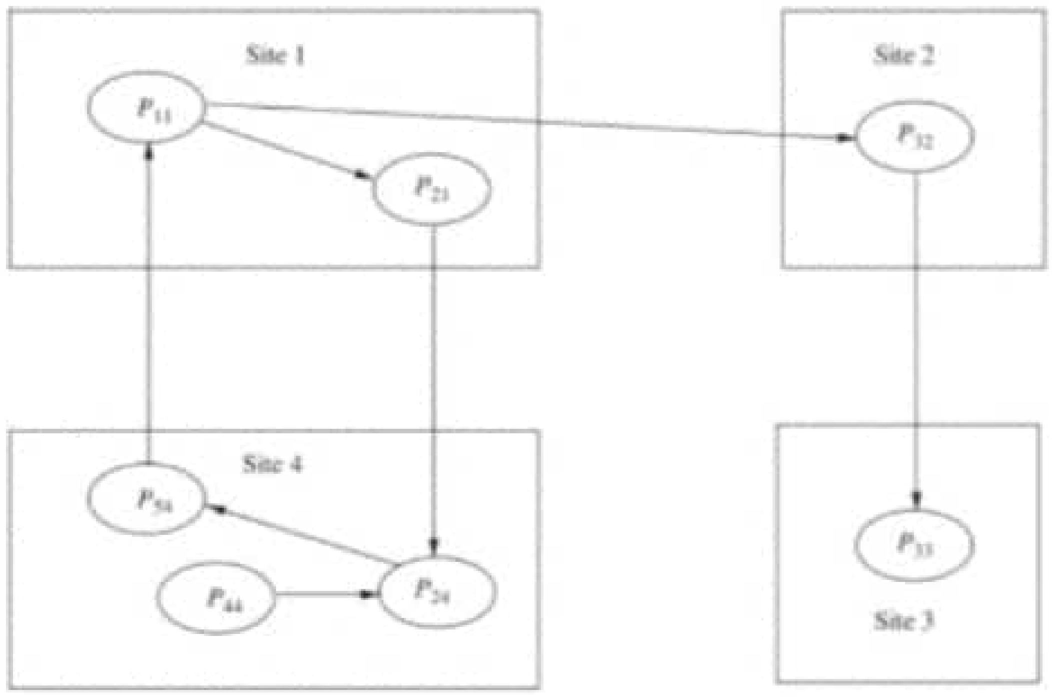
Wait-for graph (WFG)

§ In distributed systems, the state of the system can be modeled by directed graph, called a wait-for graph (WFG).

§ In a WFG, nodes are processes and there is a directed edge from node P1 to node P2 if P1 is blocked and is waiting for P2 to release some resource.

§ A system is deadlocked if and only if there exists a directed cycle or knot in the WFG

Figure 10.1 Example of a WFG.



MODULE 4

1. State the Advantages of Distributed Shared memory.

DSM has the following advantages:

1. Communication across the network is achieved by the read/write abstraction that simplifies the task of programmers.

2. A single address space is provided, thereby providing the possibility of avoiding data movement across multiple address spaces, and simplifying passing-by-reference and passing complex data structures containing pointers.

3. If a block of data needs to be moved, the system can exploit locality of reference to reduce the communication overhead.

4. DSM is often cheaper than using dedicated multiprocessor systems, because it uses simpler software interfaces and off-the-shelf hardware.

5. There is no bottleneck presented by a single memory access bus

6. DSM effectively provides a large (virtual) main memory.

7. DSM provides portability of programs written using DSM. This portability arises due to a common DSM programming interface, which is independent of the operating system and other low-level system characteristics.

The no-orphans consistency condition

Let *e* be a non-deterministic event that occurs at process *p*. We define the following:

- *Depend*(*e*): the set of processes that are affected by a non-deterministic event *e*. This set consists of *p*, and any process whose state depends on the event *e* according to Lamport's *happened before* relation.
- *Log*(*e*): the set of processes that have logged a copy of *e*'s determinant in their volatile memory.
- *Stable*(*e*): a predicate that is true if *e*'s determinant is logged on the stable storage.

Suppose a set of processes Ψ crashes. A process *p* in Ψ becomes an orphan when *p* itself does not fail and *p*'s state depends on the execution of a nondeterministic event *e* whose determinant cannot be recovered from the stable storage or from the volatile memory of a surviving process. Formally, it can be stated as follows:

$$\forall (e): \neg \mathbf{Stable(e)} \Rightarrow \mathbf{Depend(e)} \subseteq \mathbf{Log(e)}$$

This property is called the *always-no-orphans* condition. It states that if any surviving process depends on an event *e*, then either event *e* is logged on the stable storage, or the process has a copy of the determinant of event *e*. If neither condition is true, then the process is an orphan because it depends on an event *e* that cannot be generated during recovery since its determinant is lost.

Ques 5) Discuss about the SUN NFS file system. Also discuss about its architecture.

Or

What are the feature of SUN NFS file system?

Ans: SUN NFS File System

NFS was developed by SUN microsystems for use its Unix based work stations. So NFS is an extension to Unix file systems, i.e. it is transparent to Unix applications. But NFS has an open specification of file service. This has been implemented by different system like windows.

As on today, Sun's NFS is a widespread distributed file system in use and is popular. Some significant **features** of this distributed file system are:

1) Any machine can be a client and/or a server.
2) NFS is made to support diskless workstations.
3) Even if client and server have different hardware on different operating system platforms. NFS supports them.
4) NFS provides access transparency. There is no migration transparency in NFS
5) Performance rate is high (caching at the client is used)
6) Remote accessing is made as simple as local access through caching and read–ahead.
7) Remote files are accessed through normal system calls.

8) NFS is stateless. UDP is used as a transport. If a server fails, the client retries.

9) The two protocols used by the NFS client–server communication over remote procedure calls are:

   i) **Mounting Protocol:** It is used for access requests to an exported directory and

   ii) **Directory and File Access Protocol:** It is used for accessing the files and directories.

NFS Architecture

The NFS has the option of having any of the two model types:

1) **Remote Access Model:** In the remote model, the file system lies down on the server machine. The client interacts with the server via interface. It sends a request to the server regarding various file operations on a particular file. These options are provided in the interface only.

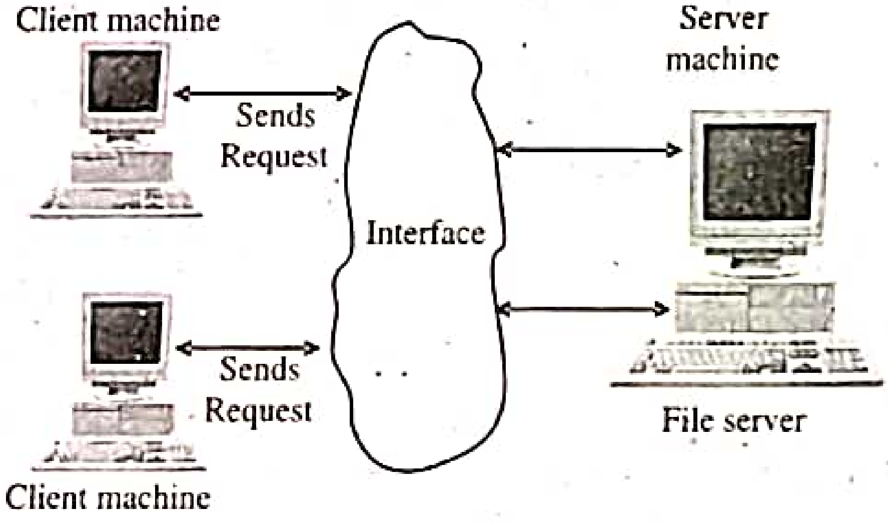


Figure 4.1: Remote Access Model

Depending on the clients request, sever performs the operations on the file. The file remains on the server only. The implementation details of various operations on the file lies with the server only. Since clients perform various operations on the remote machines (server) file, this model is called as **remote access model.**

2) **Upload/Download Model:** In the upload/download model, the client process may request for a certain file lying on the server machine. On the request, the server sends the file to client.

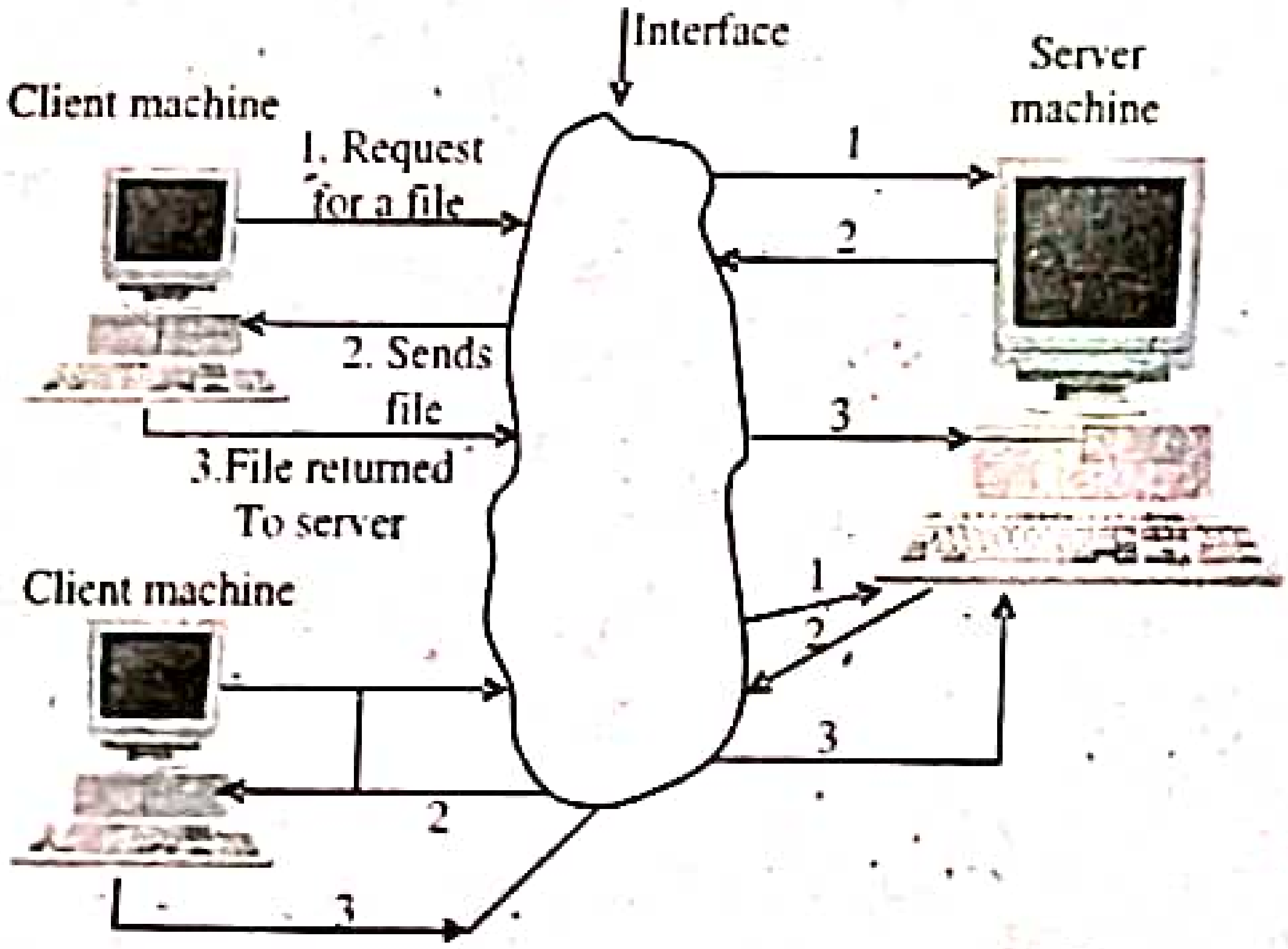


Figure 4.2: Upload/Download Model

The Client copies the file to its local disk and performs the required operations on the file. After its use, the file is returned back to the server machine. The implementation details of the file are with the client machine. The server has two versions of files available with it, the old file which was requested by the client and the new file which is being modified by the client.

Since the file is downloaded by the client before carrying out any operations on it and uploaded back to server after carrying out various operations, this type of model is called as **upload/download model.**

Though NFS has been implemented for many operating systems, UNIX based version is the most popular one.

6. Explain / compare various models of deadlocks. / Explain any different models of deadlock

Models of deadlocks

Distributed systems allow many kinds of resource requests.

A process might require a single resource or a combination of resources for its execution

Models of deadlocks introduces a hierarchy of request models starting with very restricted forms to the ones with no restrictions

1. **The single-resource model**

The single-resource model is the simplest resource model in a distributed system, where a process can have at most one outstanding request for only one unit of a resource.

Since the maximum out-degree of a node in a WFG for the single resource model can be 1, the presence of a cycle in the WFG shall indicate that there is a deadlock

2. The AND model

- In the AND model, a process can request more than one resource simultaneously and the request is satisfied only after all the requested resources are granted to the process.
- The requested resources may exist at different locations.
- The out degree of a node in the WFG for AND model can be more than 1.
- The presence of a cycle in the WFG indicates a deadlock in the AND model.

3. The OR model

In the OR model, a process can make a request for numerous resources simultaneously and the request is satisfied if any one of the reques' [illegible] 'rce' is granted.

The requested resources may exist at different locations.

If all requests in the WFG are OR requests, then the nodes are called OR nodes.

Presence of a cycle in the WFG of an OR model does not imply a deadlock in the OR model.

3. The AND-OR model

A generalization of the previous two models (OR model and AND model) is the AND-OR model.

In the AND-OR model, a request may specify any combination of and and or in the resource request.

For example, in the ANDOR model, a request for multiple resources can be of

3. The OR model

In the OR model, a process can make a request for numerous resources simultaneously and the request is satisfied if any one of the requested resources is granted.

The requested resources may exist at different locations.

If all requests in the WFG are OR requests, then the nodes are called OR nodes.

Presence of a cycle in the WFG of an OR model does not imply a deadlock in the OR model.

3. The AND-OR model

A generalization of the previous two models (OR model and AND model) is the AND-OR model.

In the AND-OR model, a request may specify any combination of and and or in the resource request.

For example, in the ANDOR model, a request for multiple resources can be of the form x and (y or z).

4. **The $\binom{p}{q}$ model**

Another form of the AND-OR model is the (p q)model (called the P-out-of-Q model), which allows a request to obtain any k available resources from a pool of n resources.

Both the models are the same in expressive power.

model lends itself to a much more compact formation of a request

Every request in the model can be expressed in the AND-OR model and vice-versa

5. Unrestricted model

In the unrestricted model, no assumptions are made regarding the underlying structure of resource requests.

Only one assumption that the deadlock is stable is made and hence it is the most general model.

This model helps separate concerns: Concerns about properties of the problem (stability and deadlock) are separated from underlying distributed systems computations (e.g., message passing versus synchronous communication).

2. Illustrate Suzuki–Kasamis broadcast algorithm

§ In Suzuki–Kasamis algorithm if a site that wants to enter the CS does not have the token, it broadcasts a REQUEST message for the token to all other sites.

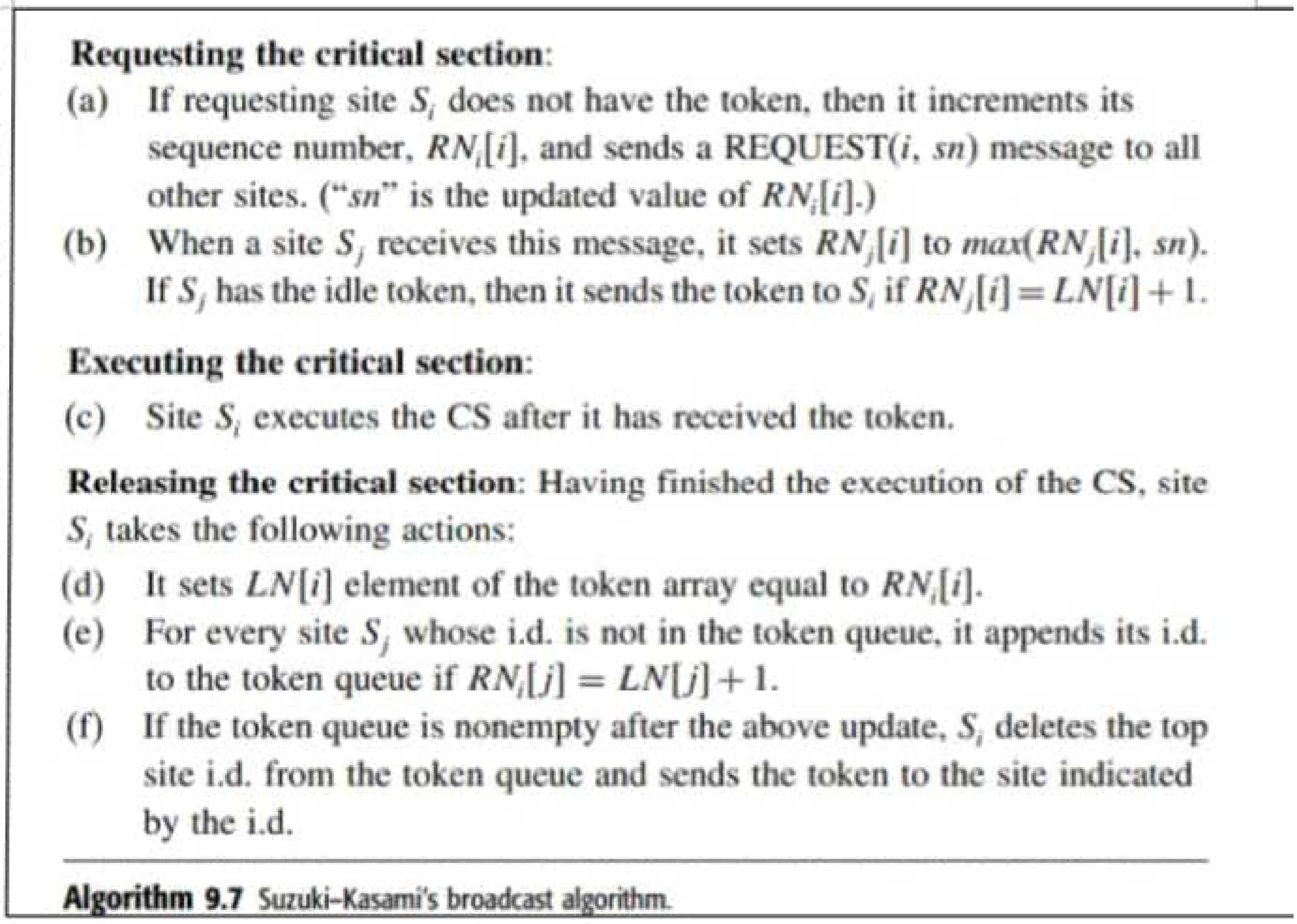
§ A site that possesses the token sends it to the requesting site upon the receipt of its REQUEST message.

§ If a site receives a REQUEST message when it is executing the CS, it sends the token only after it has completed the execution of the CS

§ Although the basic idea underlying this algorithm may sound rather simple, there are two design issues that must be efficiently addressed:

§ 1. How to distinguishing an outdated REQUEST message from a current REQUEST message

§ 2. How to determine which site has an outstanding request for the CS



Requesting the critical section:

(a) If requesting site S_i does not have the token, then it increments its sequence number, $RN_i[i]$, and sends a REQUEST(i, sn) message to all other sites. ("sn" is the updated value of $RN_i[i]$.)

(b) When a site S_j receives this message, it sets $RN_j[i]$ to $max(RN_j[i], sn)$. If S_j has the idle token, then it sends the token to S_i if $RN_j[i] = LN[i] + 1$.

Executing the critical section:

(c) Site S_i executes the CS after it has received the token.

Releasing the critical section: Having finished the execution of the CS, site S_i takes the following actions:

(d) It sets $LN[i]$ element of the token array equal to $RN_i[i]$.

(e) For every site S_j whose i.d. is not in the token queue, it appends its i.d. to the token queue if $RN_i[j] = LN[j] + 1$.

(f) If the token queue is nonempty after the above update, S_i deletes the top site i.d. from the token queue and sends the token to the site indicated by the i.d.

Algorithm 9.7 Suzuki-Kasami's broadcast algorithm.

1.Assumptions in consensus and agreement algorithm

1.Synchronous/Asynchronous communication

Asynchronous system can't detect non-arrival of message.

Synchronous system message has not been recognised by intended receipient.

2.Network Connectivity

The system has full logical connectivity.ie;each process can communicate by direct message passing.

3.Sender Identification

A process that receives message knows the identity of sender process.

4.Channel Reliability

The channels are reliable and only the process may fail.

2. Explain Lamports bakery algorithm

Lamport proposed the classical bakery algorithm for n-process mutual exclusion in shared memory systems

The algorithm is so called because it mimics the actions that customers follow in a bakery store.

A process wanting to enter the critical section picks a token number that is one greater than the elements in the array

Processes enter the critical section in the increasing order of the token numbers.

In case of concurrent accesses to choosing by multiple processes, the processes may have the same token number.

In this case, a unique lexicographic order is defined on the tuple < token , pid>, and this dictates the order in which processes enter the critical section

The algorithm can be shown to satisfy the three requirements of the critical section problem: (i) mutual exclusion, (ii) bounded waiting, and (iii) progress.

In the entry section, a process chooses a timestamp for itself, and resets it to 0 in the exit section.

In lines 1a—1c each process chooses a timestamp for itself, as the max of the latest timestamps of all processes, plus one

These steps are non-atomic; thus multiple processes could be choosing timestamps in overlapping durations.

In the entry section, a process chooses a timestamp for itself, and resets it to 0 in the exit section.

each process chooses a timestamp for itself, as the max of the latest timestamps of all processes, plus one

These steps are non-atomic; thus multiple processes could be choosing timestamps in overlapping durations.

```
(shared vars)
boolean: $choosing[1 \dots n]$;
integer: $timestamp[1 \dots n]$;
repeat
(1)    $P_i$ executes the following for the entry section:
(1a)   $choosing[i] \longleftarrow 1$;
(1b)   $timestamp[i] \longleftarrow \max_{k \in [1 \dots n]}(timestamp[k]) + 1$;
(1c)   $choosing[i] \longleftarrow 0$;
(1d)   for $count = 1$ to $n$ do
(1e)           while $choosing[count]$ do no-op;
(1f)           while $timestamp[count] \neq 0$ and $(timestamp[count], count)$
                  $< (timestamp[i], i)$ do
(1g)                   no-op.
(2)    $P_i$ executes the critical section (CS) after the entry section
(3)    $P_i$ executes the following exit section after the CS:
(3a)   $timestamp[i] \longleftarrow 0$.
(4)    $P_i$ executes the remainder section after the exit section
until false;
```

Algorithm 12.5 Lamport's *n*-process bakery algorithm for shared memory mutual exclusion. Code shown is for process *Pi*, $1 \leq i \leq n$.