Homework 3

Vishal Mehra
(vishal@npac.syr.edu)

1. High Speed Switches:-
(a). There are two major classes of high speed switches as proposed by Newman (based on switch structures) -

- Time Division based architecture
- Space Division based architecture

Time division switches can be further classified into Shared Memory switches and Shared Medium switches, whereas those employing Space division can be divided into single path and multiple path networks.

N - no. of input ports and no. of output ports
V - Transfer rate of each input and output (bit/s)

Comparison of the two classes based on:-

1. Bandwidth:

Time Division Architecture Switches: The shared memory switch suffers a bottleneck in the bandwidth of the dual ported memory and its control circuitry. The bandwidth of the dual - ported memory must be large enough to allow data transfers of NV b ps for both i/p and o/p to give a total requirement of 2NV bps in order to avoid packet loss.

In the shared medium switch since each output port has its own storage memory , the bandwidth requirements on memory are less severe and reduced to NV bps in order to avoid packet loss.

Space Division Architecture Switches: In a space division switch, concurrent paths are established from the inputs to the outputs, each path with a data rate of V bps. Thus each memory component runs no higher than 2V bps.

In knockout switch, this is further reduced to LV bps.

2. Blocking Characteristics:

Time Division Architecture Switches: Both the shared memory and shared medium switches are inherently free from internal

blocking, but both suffer from output blocking. To remedy this, output buffers are provided which is the shared memory itself in case
of shared memory switch, and FIFO memories in case of shared medium switch.

Space Division Architecture Switches:All space division architecture switches suffer to varying degrees, from internal blocking. However not all space division switches suffer from output blocking, for example, the Crossbar switch (congestion at the outp
ut is not possible for this switch).

3. Buffering Requirements:

Buffers are needed to alleviate the problem of blocking.

Time Division Architecture Switches: No input buffers are required because of absence of internal blocking. The shared memory and FIFO memories respectively act as output buffers for the shared memory switch and shared medium switch. The shared memory ac
ts as N virtual ouput FIFO memories. The required size of each buffer is derived from the desired packet loss probability for the switch.

Space Division Architecture Switches:In fabrics exhibiting internal blocking, it is not possible to buffer packets at the output, as is possible in TDA switches. Instead buffers are located at places where potential conflicts along paths occur, or upstre
am to them. Ultimately buffers may be placed at the inputs of the switch as is the case with crossbar switch.

(b). The Knockout Switch has a Space Division architecture.Each input has a unique path to the output and as such it has a single path network.Thus it is able to (almost) eliminate the problem of internal blocking. This feature sets it apart from the othe
r space division switches as all suffer from internal blocking.

Secondly, it has a very low memory bandwidth requirement of only LV bps. If per chance L+1 packets arrive at the input of the NxL concentrator there is a possibility of packet loss and hence internal blocking. However this can also be conceivably reduce
d by the provision of buffers at the NxL concentrator input.

(c). Significance of the Batcher portion of the Batcher-Banyan Switching Network:

Banyan type switches suffer from internal blocking and consequent reduction in throughput. One solution to improve throughput is to sort the input packets in order to remove output conflicts, and then present to the Banyan router packet permutations whic
h are guaranteed not to block. This is the function (and significance) of the Batcher portion of the Batcher-Banyan switching network.

The Batcher sorter sorts the incoming packets according to their requested output addresses and removes packets with conflicting output addresses. This is done with a "Trap network". Remaining packets are concentrated to the top lines of the fabric. This
is accomplished via a reverse Omega network. Concentrated packets are then routed via the Omega network. Packets which are not selected by the Trap network are recirculated, being fed back into the fabric at later slots.

Thus with the batcher sorter, throughput is considerably improved.

2.1

Functions of Host-to-Network Interfaces

1. Protocol Processing: The protocol processing includes packetization/ depackietization, error handling, flow/ congestion control,
and routing.

o Packetization / Depacketization

In order to trnsmit data over a network, the data need to be packetized
into pieces and attached with protocol processing headers and trailers to
from data packets according to different communications protocols. At the
receiving side, the arrived packets will be reassembled into the origional
message by striping off the headers and the trailers. The process f reassembly is referred to as depacketization.

o Error Handling

The Transport layer is responsible for providing reliable data delivery to the user. Therefore transport layer protocol processing includes detecting and correcting errored packets, since data may be lost due to congestion overflowm and it may be reordered or duplicated as a part of processing.

o Flow/congestion control

To protect both the network and the receiver, the sender must be regulated not to send data at a faster rate than that can be handled by the network and/or the receiver. The minimal in-band control function involves the pacing of the data at the transmitter and the monitoring of arrivals at the receeiver. The actual computation and negotiation of the transfer rate can performed on an out-of-bod basis

2. Data Manipulation

There are six manipulation functions that have been founded in most protocols.

o Moving to/from the network

Moving data to/from the network involves some sort of serial-to-parallel transformation. This function is uusually performed in custom hardware which decouples the timing of the network from that of the host.

o Error Detection

Transmission errors are detected by calculating a checksum over the data on both the transmission and the reception side. Checksum is computed by reading the data

o Buffering for retransmission

At the sending side of a transfer, many protocols reserve a copy of the
dat, so that, when transmission error occurs, the data can be resent.

o Encryption / Decryption

Encryption/ decryption may be needed to achieve security purpose at
sending and receiving side

o Moving data between user space and interface

The networked data may not be moved directly between user address space
and network interface. It is usually moved to some intermediate buffer in
the host kernel buffer, which is dictated by the structure of the I/O
system.

o Presentation formatting

Most data transfers require that the data be reformatted into some common
or external data representation.

On one extreme, these functions can all be implemented at the host which
will result in simple inexpensive host-network interface. Another extreme
is to implement all the functions on the interface. Most existing
host-to-network interfaces are implemented in trade-offs between these two
exteremes.

2.2

o Medusa FDDI Interface

The Medusa FDDI interface,loosely based on WITLESS, is a research prototype that was deigned for the HP Apollo Series 700 workstations at
Hew-Packard Labs in Bristol. The interface contains the retransmission
bufer memory that is required to support a single-copy protocol stack., and
appears to the host as block of memory in I/O space. All network, transport, and socket layer processing is performed by the host. And the
interface buffer is used as an mbuf in convetional protocol processing,
resulting in a single copy architecture.

A user process presents data to the interface buffer by means of socket
layer procedures. By allowing the socket layer to perform the copy operation, the data is packetized before passing to TCP. And a checksum is
copmuted on the network buffer memory, then the device driver copied the
headers into the space reserved at the front of the buffer and causes the
packet to be transmitted.

o UPenn ATM interface

The host interface designed at Unoversity of Pennsylvania has been centred on developing at high-performance host interface for IBM RISC
System /6000 workstation host in Aurora Gigabit Testbed Environmnet

The Design philosophy for teh architecture is based on providing a "common
denominator" set of servcies in dedicated hardware. All per-cell activities,
such as ATM header and adaption layer creation and processing(including
segmentation and reassembly) are performed by the host interface in hardware. The host is responsible for all high level activities to achieve

flexibility in protocol implementations.

The MicroChannel architectures bus on RISC System/6000 has been chosen as the host interface's point of attachment for its high bandwidth. DMA scheme is used in transferring data between the memory in host and the buffer on interface.

It has on-interface hardware elements for cell related activities and DMA device for data transfer. Two data copying are involved. The network interface is connected to the host main memory through I/O bus.

3 (i). The main functions of a transport protocol are the following:

1. Packet Assembly:
   The different network protocols have associated with them a limit on the size of the data field. For example for Ethernets this limit is 1500 bytes and for ATM this is 9416 bytes. This rquires that all messages to be delivered need to be broken down into these packet sizes that are less than the Maximum Transfer Unit (MTU) of the network layer. The task of the transport layer is to divide these messages into packets before transmission and then reassembling them on receiving them at the receiving computer.

2. Addressing:
   The transport layer's task is also to provide a network-independent message transport service between pairs of network ports. These ports are local to a computer. The transport address is composed of the network address of a host computer and a port number.

(ii) 1. Packet Assembly:

The MTU for IP is 64 kB whereas the MTU for TCP is unlimited. TCP therefore decomposes the data stream into chunks of data and constructs IP packets that are at most 64kB in length. It then attaches its header and carries out its checksum on the data. IP next attaches its own header to the frame before passing it to data link layer. The network interface layer next divides the frame into smaller sizes to fit the MTU of the underlying network. On receipt of a frame, the IP layer strips off its header and carries out its checksum on the received frame. It then passes the frame to the TCP layer which strips off its header, carries out its checksum and assembles the message from the received IP frames.

2. Addressing:

In TCP/IP networks each host computer has several ports associated with it. Many of the standard internet services such as rlogin, ftp, etc., are run on well known ports. These are reserved ports and are known to all computers. In addition there are sevaral ports available for general use. To access a sercice at a given host, a request is sent to the relevant port on the host.

(iii) When the network transfer rates increase to the Gigabit/second range then the TCP/IP execution stack is not able to cope with the data rate to do the checksumming and packet assembly functions. The CPU becomes the bottleneck as it is not able to provide the processing power to cope with the high data rates. Checksumming is a major overhead in TCP/IP and requires high

CPU usage. Further, data copies are made once from the network card to kernel space and from there to user space. This is also a major overhead as it requires byte copying of the data from the different layers. The TCP/IP execution stack therefore becomes a bottleneck in Gigabit networks.

(iv) In high speed communication protocols the issues outlined in (iii) above are addressed in several ways. First, memory mapping is used to map the user space directly onto the network card thus avoiding the extra copy to kernel space. This leads to substantial speedups. In addition, checksumming is done while the data copy is being carried out leading to additional speedups. In the UDP transport protocol checksumming is not carried out and thus leads to higher data rates with this protocol. NETBLT was developed for high throughput bulk transfer rates and uses a large MTU and transmission of several of them may be concurrently active. Larger MTU leads to fewer packet assembly being performed leading to savings in this area. Further the checksumming can be performed once on the entire block instead of on multiple smaller blocks as in TCP. This further reduces the overhead associated with this task.

(4)

An OS allocates computer resources such as memory, I/O devices and files to processes. The OS also provides an interface to the user by the means of a command line interface to invoke OS utilities like editors etc. In a centralized OS, all these tasks are achieved by a single monolithic kernel.
Hence the OS caters to user command management, memory management,

CPU management and I/O management. The result of this is that the OS kernel can grow to be quite large resulting in an inefficient implementation which could be buggy and raise costs to debug and maintain.

The distributed OS also has to perform the same duties as a centralised OS does but its job is more difficult. In order to achieve execution of a given task, a distributed OS may have to exploit resources which could be geographically dispersed. Hence it needs to coordinate the usage of these resources in an efficient and correct manner. The distributed OS (DOS) thus has to provide transparency to the user. The user sees the DOS as a single processor operating system. In addition, a DOS must also take into account user authentication and user authorization. This is necessary to provide a secure distributed environement for computing. Multiple techniques are used to accomplish this. In Amoeba this is accompished by use of capabilities.
In order to obtain a global state of the system in a shared memory system (such as a centralised system) is trivial. However, for DOS, this poses a problem as there is a lack of a global memory and a global clock to synchronize transactions. There is also the unpredictability of variable message delays which exacerbates this problem.

Traditionally, the centralised OS contains a large monolithic kernel and all the tasks are accomplished by trapping to the kernel which actually performs the requested task and passes the result back to the user. Most of the DOS that are emerging are micro kernel oriented. Here the kernel itself is relatively small and performs basic memory, and I/O management. Most of the functionality of the OS is implemented via user processes

(which could be
running on a remote machine). Here functionality such as file I/O
etc is
managed by the user process which makes use of the micro kernel to
complete
its task.

Thus, both the centralised and distributed OS have to
provide similar
capabilities to the end user. DOS are a relatively new development
and they
aim to exploit the parallelism available from multiple CPUs
available to them
to accomplish tasks more efficiently. However, the issues involved
in providing
these capabilities are drastically different between the two
systems. DOS
infact aims to portray itself as a single processor system by
incorporating
various types of transparency (Access, location, concurrency etc).
From avneesh@nova.npac.syr.edu Thu Nov 9 19:09:35 1995
Date: Thu, 9 Nov 1995 03:32:13 -0500 (EST)
From: Avneesh Pant <avneesh@nova.npac.syr.edu>
To: vishal <vishal@nova.npac.syr.edu>,
    Animesh Chatterjee <animesh@nova.npac.syr.edu>,
djaising@cat.syr.edu
Subject: q5. Hw3

(5)
Distributed Operating Systems

Design of Distributed Operating Systems

The design of a DOS can be characterized in terms of the techniques
used to implement the following

Interprocess Communications:
IPC design affects the design of applications as well as the
performance of the system. IPC is usually carried out using message
passing and thus the IPC performance and reliability depends
heavily
on the underlying communication system. The most common message
passing model used is client-server model. There can be systems
with
blocking or non-blocking communication primitives. Also blocking
could be reliable and unreliable. Also buffering of messages is am

important decision. So the main decisions to be made are blocking, reliability and buffering.

Resource Management:
Resource Management is concerned with making both local and remote resources available to the user in an efficient and transparent manner. Some methods for processor allocation, scheduling and load balancing need to be developed. Load balancing techiniques can be achieved by either migrating data, computation or process.

Fault Tolerance:
Fault tolerance is the most significant aspect of distributed computing. Many systems achieve fault tolerance by duplication os the system, resources etc. However the most difficult problem is to discover failures and recover from them. Also there are different categories of failures. Techniques used for making the system fault tolerant include atomic transactions and redundancy techniques. Atomic transactions refer to the ability of a system to reliably access a shared database in the event of failures. Redundancy Techiniques use process replication to combat hardware failures. Another version of redundancy combats the problems
arising due to programming error itself.

System Level Services:
Systems level services include :-

File Service
A common file system for a distributed systems includes file servers.
This type of service reduces costs by elimination of many disk drives
as the file services can be accessed via diskless workstations. The concept
is that each user has access to reasonable amount of memory that is dedicated
solely to the user. The user had the ability to control the access to his
own files. The use of file servers results in the ability to have many users sharing files without constantly swapping disks and updating. Important issues in distributed file systems include atomic updates of files, transparency, availability, concurren cy control, directory service.

Naming Service

The distributed systems is responsible for supporting the management and protection of objects for access to files, directories, mailboxes, processors, and I/O devices. The naming scheme followed by most implemented systems is that of mapping the name giv
en in a system call to the device implementing the desired system call. Naming is basically a problem of mapping between two domain. Mapping may be completed in a single step or be multi step. It also may not be unique.

Security and Protection Service

The distributed system is responsible for the security of the system as a whole and the protection of resources in the system. Security and protection are necessary to avoid unintentional or malicious attempts to harm the integrity of the system. Two issu es needed to be addressed are authorization and authentication. Authentication is making sure an entity is what it claims to be. Authorization is granting a process access to do something based on the previleges it has.

Techniques Employed in Distributed Operating Systems

LOCUS

The assets of LOCUS are its highly transparent file system and slightly less
transparent process-execution location and exhaustive fault tolerant system.

[File System:]

The system architecture of LOCUS emphasizes the transparency of the network,
especially where files and their naming is concerned. It provides a single
tree structure for file-naming with all objects in the file system on any
machine in the cluster in a single tree. LOCUS allows file replication by
allowing a logical file group to exist in any of several physical containers.
Each container holds only a subset of the files for a given subtree. A file's
name is then constructed from a logical filegroup number and a file

descriptor number. Also to maintain consistency a version vector is associated with each file. This ensures that all replicated files in the
system are consistent.
There are 3 types of logical sites in LOCUS : Using Site (US), Storage Site
(SS) and Current Synchronization Site (CSS). The US is the requesting site
which will receive the file pages, SS has the requested file pages, CSS acts
as the arbiter in following the global access policies and selection an SS.
Due to file replication, file creation and deletion have added resposibilities. At creation the system must know how many files to create
and where to store the replicated files. This is achieved through the
inherited variables and system calls. A file is
deleted using the same mechanisms used to update files.

[System Structure:]
As with the file system, LOCUS strives to provide transparent support for
remote processes. The site selection mechanism allows for executing software
as either a local or remote site with no change required for software. The
user of a process determines its execution location through information
associated with the calling process or shell commands. The UNIX commands fork
and exec are used to implement local and remote processes.
LOCUS supports explicit functions, pipes ans signal, and implicate mechanisms
thru the use of a token mechanism. The token marks which resource is valid
and resource access requires a token.

[Fault Tolerance:]
LOCUS heavily emphasizes fault tolerance. Two major issues involved are
recovery and reconfiguration. Recovery deals with maintaining synchronization
among files within a partition. Once partitions are merged conflicts must be
resolved in the two partitions through automatic or manual means.

Reconfiguration allows LOCUS to transparently reconfigure the network
and system as necessary due to faults. It is essential during reconfiguration to minimize message overhead to maintain transparency.

AMOEBA

Amoeba was built to support distributed computing across a computer network
and it provides an emulation facility which serves to bridge the gap between
Amoeba and existing systems. The chief goals were transparency, single system
wide file system, software based on objects, security and protection,
interprocess communication and management.

[Hardware Architecture:]

The Amoeba architecture consists of four principle components:

Workstations which are allocated one per user and do all the fast interactive
tasks like CAD/CAM applications.

Processor pools from which a group of CPU's can be dynamically allocated as
needed mainly for compute intensive tasks.

Specialized servers such as file , database, directory servers which do
dedicated tasks.

Gateways which are used to link Amoeba systems all over the world into a single
uniform system.

[Software Architecture:]

Amoeba is an object oriented distributed operating system. A client process
sends a request message to the server process that manages the object. and
blocks until it receives a reply. To handle multiple simultaneous requests, a
process can be sub-divided into lightweight processes called

threads. By
having a thread for each request, a server process can handle multiple requests
simultaneously. Threads are non-preemptive. Objects are named and protected
by capabilities. These capabilities combined with transactions provide a
uniform interface to all objects.

[Interprocess Communication:]
To perform an operation on an object, a client sends a request to the server
which manages the object. This is done using Remote Procedure Calls.
The system provides 3 basic system calls for RPC's :
getrequest(reqheader,reqbuffer,reqsize)
dooperation(reqheader,reqbuffer,reqsize,repheader,repbuffer,repsize)
sendreply(repheader,repbuffer,repsize)
When the server is ready to accept request from clients, it executes a
getrequest system call, which causes it to block. In the call, it specifies
the port on which it is willing to receive requests. When a request arrives,
the server unblocks, performs the work using the parameters and sends back the
reply using sendreply system call. RPC's and object manipulation
can be specified using AIL (Amoeba Interface Language).

[Process Management:]
In Amoeba the key concepts are segments and process descriptors. A segment is
a contiguous chunk of memory which has code or data and read/write operations
can be performed on it by its holder. A process descriptor is a data structure
that provides information about the process:
the requirements of the system where the process can run like the class of
machines, memory, instructions.
Address space layout which includes no of segments, size of segment, virtual
address.
The state of each thread of control like stack pointer, program counter, PSW

and registers.
The list of ports for which the process is a server.
A process is created by executing the following steps:

Get the process descriptor for the binary from the file system.
Create local segment or a file and initialize it to the initial environment of the new process. the environment consists of a set of named capabilities,
and the arguments to the process.
Modify the process descriptor to mark the first segment just created.
Send the process descriptor to the machine where it will be executed.

The memory server extracts the capabilities for the remote segments and
fetches the code and the data segments from wherever they reside by using the
capabilities.

[File System:]
It consists of a highly unusual service called the Bullet service which
stores files contiguously both on disk and memory. When the server is booted,
the entire i-node table is read into memory in a single disk operation and
kept
there while the server is running.. When a file operation is requested, the
object number field in the capability is extracted, which is an index into the table. The entry thus located gives the disk address as well as the cache
address of the contiguous file system.
The Bullet service provides three operations : read file, create file and
delete file. The files are immutable which facilitates retrieval, administration, replication and caching.

[Directory Service:]
The directory in Amoeba has a hierarchichal structure , which facilitates the
implementation of partially shared named spaces. Directories in Amoeba are also
treated as objects and users need capabilities to access them. Thus a

directory capability is a capability for many other capabilities. The basic operations that the directory supports are lookup, enter and delete. It also supports replication of objects. When a new object is entered into the directory, the directory service asks the object manager to supply multiple copies of the object, which are then physically distributed and their capabilities entered into the directory.

[Transparency:]
Transparency was one of the guiding design issues in Amoeba. When a user logs in, they log into the system as a whole and not into one particular machine. The user is not aware of where their files and data are stored, or on which machine programs are running. Processes can be migrated transparently and run on any computer which meets the hardware and software requirements of the process.

[Security and Protection:]
In Amoeba, objects are protected using capabilities ans server access is controlled by ports. Access to objects like files, directories, I/O devices can be given or denied to users by specifying these access rights in the capabilities. The capabilities are encrypted using a key chosen form a large address space. This key is stored in the check field of the capability itself.
Amoeba also provides a method for secure communications. A function box (F box) is put between each computer and the network which may be implemented in hardware as a VLSI chip on the network interface board or in software in the operating system.

V

The V system is built on he philosophy that high-performance communication is the most critical facility fora distributed system and the system is defined not by the software but by the protocols it implements.

The V system uses a transport protocol called VMTP, that has been optimized for request-response behaviour. It has implemented multicast for group communications.

Processes in V are grouped together under a single group identifier. V does not provide any specific reliability or fault tolerance techniques. Software reliability is acheived by keeping the kernel small and software modular.
Security and protection is provided by the authentication servers, though there are no specific security measures to protecet the network and the operating system.

[Hardware Architecture:]
The V-system hardware environment consists if high resolution workstations having good processing power, a large memory and virtual address space and a graphics input device, a fast communications network between workstations, Dedicated servers for printing, file storage, computation support and other services and access to special purpose computers and wide area networks.

[Software Architecture:]
The V-system is based on the client-server model. The various computers in the system communicate via message passing. Each computer in the system

runs the V-kernel, which provides the basic facilities like process and memory
management, communication and device management. The kernel provides a
``software backplane'' just like a hardware backplane, in which modules can be
plugged in. The system services are accessed by application programs through
procedural interfaces.

[Interprocess Communication:]
Three forms of interprocess communication mechanisms are provided in the V-
system. In the first one, a process may communicate with another by sending
and receiving fixed size messages. A SEND by a process blocks it until a reply
has been received by it.

In the second form of interprocess communication, a process can pass a segment
in it's team space (address space on the computer) to another process. After
sending the message, the sender is blocked, while the receiver can read or
write it's segment using the promitives, CopyFrom and CopyTo provided by the
kernel. The sender is unblocked only when it receives a reply from the other
process, after it has completed its task. This model uses the 'data segment'
provided by VMTP.

The third type of IPC provides a one-to-many coomunication facility to address
the need for group communication. V supports the notion of a process group -
a set of processes having a common group identifier. Processes, members or
non-members, can send and receive messages from process groups. To send a
message to a group, all that is required is the group identifier instead of
the process identifier as the parameter.

[Resource Management:]

The key resources that the kernel manages are processes, memory and devices,
displays, keyboards, and mice.

A process in V consists of a priority, a state, a team space and a stack. The
state is the machine state for that process and the team space is the area o
memory to which it has direct access. Each team space may contain several
processes. The kernel process sesrver provides functions to create, destroy,
query, midify and migrate a process.

The V kernel memory manager supports demand paging by associating address
spaces, called regions, to an open file. The kernel provides caching and
consistency mechanisms for these regions. The kernel also provides device
support for devices like disks, mice, and network interfaces throught the
device server. The device server is device and machine independent and
implements the UIO system interface.

[File Service:]
The file system in on dedicated machines, with the kernel on them providing
support for disks in the server routines. The storage server is a member of
a group having a well known group identifier. Processes can communicate
with particular servers using this identifier plus the V naming protocol. The
V system implements the mapped-file I/O mechenism to provide fast and
efficient access to files. A file OPEN request is mapped onto a locally
cached open file. Files are accessed using the standard UIO.

[Naming:]
The V scheme is structured on a three level is structured on a three level
model consisting of character string names, object identifiers and entity

identifiers. There is no specific name server which the processes have to
contaact to get a name to address translation done; as each object manager
impplements the naming for the set of objects it manages.

To avoid the overhead of character string name lookup each time, an object
identifier is used to refer to thee object in subsequent oerations. An
object identifiers consists os two fields the first one specifying the
identification of the manager and the second one specifying the ID of the
object on that manager.

Finally entity identifiers are fixed-length binary values which identify
transport level endpoints. They identify processes, as a process or group
is effectively associated with one port. Effectively they serve as process
and group identifiers.

[Security and Protection:]
V provides limited security protection. Network security against intruders
and tappers is not provided in V. In V, as in Unix, each user has an account
name and an associated password. A user is also given a user number.

The files and objects are protected by the servers that implement them. The
hosts are protected by the team servers. To access either a process sends a
request to an authentication server. The request includes the account name
and password of the user. The authentication server matches the name and the
password against an encrypted copy stored on it.

Clouds

Clouds is DOS based on Object Oriented Design.
Clouds relies much more on the object oriented design and based on object thread model. The objects in clouds have specific operations which define their interfaces. Access to this objects is accomplished through the defined operations. User activity is based on threads. The thread defines a logical path of execution. For portability and modularity Clouds uses specific approach which means that the functions are implemented at the highest possible level. This approach results in three level of implementation. The lowest level is the Ra kernel. The second level of implementation is system objects. The third level deals with non critical service such as naming and spooling.

[Fault Tolerance:]
Certain failures in the Clouds system can cause consistancy problems among the nodes. Therefore an atomicity mechanism is used to insure a thread either completes successfully or does not effect the system state. The invocation based consistance control provides a wide range of atomicity semantics which automatically lock and recover persistant data. To insude that a computation which completes is successful, clouds utilizes parallel execution threads. The system basically allows the thread to execute independantly on a specific number of nodes. The results are decided by quorum. To support parallel execution threads the system is required to support object replication, support computation replication, have an atomic commit mechanism.

[File System:]

The files are stored as objects which have read write invocations, which
can access and update data. Since objects are persistant the data will
exist until explicitly deleted. This method saves the additional overhead
which would be normally required to convert data from byte stream format
and back again. Also considerable efforts are saved due to translations
between Unix and Clouds. This eliminates the need to adapt programs to a new
environment.

MACH
It was intended to be compatible with UNIX and support large-scale parallel and distributed computing. Its main features include its large address space, capability to provide transparent access to network resources, capability to exploit parallelism in b
oth system and applications, and its portability to a large collection of machines.

[Process Management:]
Mach split the traditional UNIX abstraction of process into a process
and threads. A process in Mach consists of an address space and a collection of threads that execute in that address space. Processes are passive. Execution is associated with threads. Processes are used for collecting resources related to a group of cooperating threads into convenient containers. Thread creation and destruction
is performed by the kernel and involves updating kernel data structures.
All threads within a process share the virtual memory address space and
communications privileges associated with their process.

[Thread Scheduling:]
The primary data structure used by the Mach scheduler is the run queue, a
priority queue of runnable threads implemented by an array of

doubly
linked queues. A hint is maintained that indicates the probable location of the
highest priority thread. The highest thread cannot be at a higher priority than the hint, but it may be at a lower priority. This allows the
search for the highest priority thread to start from the hint, potentially avaiding the search of a dozen or more queues because the highest possible priority for most user threads is 50 to match UNIX. Each run queue also contains a mutual exclusion lock and a count of threads currently enqueued. Mach is self scheduling in that, instead of having threads assigned by a centralized dispatcher,
individual processors consult the run queues when they need a new thread to run. A processor examines the local run queue first to give bound threads absolute preference over unbound threads. If the local queue is empty, the processor examines the global run queue. In
either case, it dequeues and runs the highest priority thread.

[Interprocess Communication:]
The basis of all communication in Mach is a kernel data structure called a port which is essentially a protected mailbox. When a thread in one process wants to communicate with a thread in another process, the sending thread writes the message to the port and the receiving thread takes it out. Each port is protected to ensure that
only authorized processes can send to it and receive from it. Ports support unidirectional communicaiton, provide reliable, sequenced message streams. Ports may be grouped into port sets. A port may belong to at most one port set. It is possible to read from a port set. Mach provides special ports to communicate with the kernel. A message is a string of data prefixed by a header. Messages may be simple or non-simple. Messages are actually mapped into the virtual memory resources of the processes and hence the IPC is far more efficient than old-time UNIX implementations where the messages are copied from one process to the limited memory of the kernel and then to the process receiving the message.

[Network Massage Server:]
Communication between processes running on different machines is achieved using Network Message Servers (NMS). The NMS is a multithread process that performs functions like interfacing with local threads, forward messages over network, translate data types, network-wide name lookup service and authentication services.

[Resource Management:]
The following are discussed :

itemize
Processor Allocation
Gang scheduling, which guarantees simultaneous scheduling of an application's components, is one use for processor allocation in
mulitiprocessor OS. Mach supports multitude of applications, languages and programming models on a variety of multiprocessor architectures, flexibility is highly desirable in design of processor allocation facility. Processors only execute threads assigned to the same processor set and vice-versa and every processor and thread is always assigned to a processor set. If a processor set has no assigned processors, then threads assigned to it are suspended. Processors and processor sets are represented by Mach ports. Processor sets also have a name port for identification and status queries, but this port cannot be used to manipulate the processor set. Responsibility for the allocation and use of dedicated processors is divided among the application, server and kernel. The application controls the assignment of processes and threads to processor sets. The server controls the assignment of processors to processor sets. The kernel does whatever the server and application ask it to do.

Memory Management
Each Mach process can use upto 4 GB of virtual memory for execution of its threads. This space is used for the memory objects, messages and memory mapped files. Processes can create memory objects for use by its threads and these can be mapped onto the space of another process. Spawning new processes is efficient because memory does not need to be copied to the child. The child needs only to touch the necessary portions of its parent's address space. When spawning a child, it is possible to mark the pages to be copied or protected. Each memory
object that is mapped in a process address space must have an external memory manager that controls it. Different classes of memory objects are handled by different memory managers. To map an object into a process address space, the processs sends a message to a memory manager requesting it to do mapping. Three ports are required to do the job, object port, control port, name port. A strict protocol is used communication between the kernel and memory managers. This protocol consists of small number of messages that the kernel can send to a memory

manager and a small number of replies the memory manager can send back to the kernel. Kernel sends messages in form of asynchronous messages on object port and memeory manager sends an asynchronous reply on the control port.

Distributed Shared Memory

The Mach external memory manager concept facilitates implementation of page-based distributed shared memory. When a thread references a page that it does not have, it creates a page fault. Eventually the page is located and shipped to the faulting machine. Since Mach already has memory managers for different classes of objects, a shared page memory object is introduced. Also it is possible to have a single memory manager that handles all shared pages or to have a different manager for each shared page or a collection of shared pages. Shared page is either readable or writable and consistency control is implemented.

X-Kernel

The X-kernel is an experimental distributed operating system that aims at providing a uniform access to resources throughout a national internet network. The focus on having the domain of teh system covering a wide-area network requires that the system must support a wide range of file systems and communication protocols. The X-kernel supports a library of protocols and it can use a combination of protocols to access the required resources.

[Process And Memory Management:]

In X-kernel, the transfer between user and kernel space has been made very efficient by having the kernel execute in the same address space of the user data. The user process can access the kernel process efficiently because they run in the same address space.

[Communication Protoclos:]

There are three communications objects used by the kernel to handle the
protocols, sessions and messages. A different protocol object is used for each
protocol type being implemented. The session object is the protocol's object
interpreter and it contains the data which represent the state of a protocol.
The message object is transmitted by a protocol and it's session object. There
are multiple address spaaces in teh X-Kernel which consist of the kernel area,
user area and stack. If tehre is more than one process in a given address
space then the kernel and user area are shared but each each process as
own private stack area. To communicate between processes and in different
address spaces or different machines the protocol objects are used and
messages are sent. Processes in teh same address space can synchronize using
kerenel semaphores. A process can execute in user mode or in kernel mode.

[File System:]
The X-kernel file system allows it's user to access any resources regardless
of the user location. It provides a uniform interface by implementing a
logical file system that can contain several different physical file systems.
The logical file system allows the file system to be tailored to users'
requirements.

The file system is logical in the sense that it provides only the directory
service and relies on an existing physical file system for the storage
protocols. The Logical File System maps a file name to the location where it
can be found. Each user defines his own private file system out of the existing
physical file systems. The separation of the directory functions

from the
storage functions is achieved through the use of the private name space
protocol and the uniform file access protocol.

[Command Interpreter:]
In a similar manner to the services provided by the X-kernel logical file
system, the command interpreter provides a uniform access to a heterogenous
collection of network services. Ths interpreter is different from the services
provided by distributed operating systems in that it's services cover
resources available throughout the network, not just the local resources
provided by a local operating system.

MOSIX
MOSIX is an enhancement of UNIX which allows users to use the resources of a
NOW configuration, without any change at the application level. By using
transparent, dynamic process migration algorithms MOSIX enhances the network
services namely NFS, TCP/IP of UNIX to the process level, by supporting load
balancing and dynamic work distribution (leveling). In clusters of homogeneous
computers, the main advantages of the MOSIX extentions, vis-a-vis other multi
computer systems, are the performance of dynamic load balancing among the
computers in a transparent way and the possibility to scale up the
configuration. This means that most users see a conventional UNIX with no need
to modify any existing applications.

[Characteristics & Design Techniques of MOSIX:]
(a) File System: MOSIX uses standard NFS as its file system.
(b) Network Transparency: The network is completely invisible to the
application software.
(c) Dynamic process migration: Processes can be migrated among homogenous

processors. However, this does not exclude MOSIX from being part of a wider
hetrogeneous network.
(d) Load balancing: Process migration allows assignment and re-assignment of
processes to processors for near optimal resource utilisation.
(e) Dynamic configuration: Computers may be added and removed with minimal
side-effects.
(f) Fault Tolerance: Increased availability is provided by isolation of faults.
(g) Replicated Kernel & Resources: The system is replicated in each node.
(h) Scalability: The configuration can be scaled up to a large number of nodes.
(i) Compatibility: MOSIX is compatible with BSD like UNIX.