COMP90015 Distributed Systems

1_Introduction to Distributed Systems and Characterisation

1. Defining Distributed Systems: a distributed system is a collection of independent computers that appear to the users of the system as a single computer.

2. Briefly explain the difference between a computer network and a distributed system
Networks: Is a collection of spatially separated, interconnected computers that exchange messages based on specific protocols. Computers are addressed by IP addresses.
Distributed System: Multiple computers on the network working together as a system. The spatial separation of computers and communication aspects are hidden from users.

3. List three reasons for using a distributed system
Functional Separation: Modularity, 模块化, 不同计算机不同作用
Reliability: Long term preservation and data backup (replication) at different locations.
Economies: Share resources, sharing a printer, reduce the cost of ownership.

The main motivation to use DS is **Resource Sharing**

- Hardware Resources (Disks, printers, scanners etc.), Software Resources (Files, databases etc), Other (Processing power, memory, bandwidth)

Inherent distribution: 本来就分布， 如不同人管理维护不同information, supermarket chains
Power imbalance and load variation: We are forced to use DS as one system cannot handle!

4. Consequences of Distributed Systems
Concurrency: Tasks happen independently, but may need to exchange status update or other information.
Heterogeneity (异质性): Different hardware, different software, different protocols
No global clock: We cannot assume correctness of timing given.
Independent Failures: Many components working together (ie goog), any can fail

后面是workshop没有的
5. Characteristics of Distributed Systems:
Parallel activities, Communication via message passing (no shared memory), Resource sharing, No global state, No global clock

6. Goals of distributed systems:
Connecting users and resources, Transparency (User should be unaware of complexity behind system), Openness (Use standards so anyone can interface with, understand and extend), Scalability (can add capacity to deal with growth), Enhanced Availability

以下Tutorial 4
7. Challenges of distributed systems:

Heterogeneity异质性: heterogeneous components must be able to interoperate互操作 across different OS, hardware architectures, programming languages…

- Where?
  - Networks ○ Computer hardware ○ Operating systems
  - Programming Languages ○ Implementation by different developers
- How to deal with it?
  - Standard protocols ○ Middleware ○ Portable code
  - Adhering to articulated APIs, message formats, and data types

Mobile code is sent from one computer to the anther to run at the destination (e.g. Java applets):

- Code that is compiled to run in one OS does not run in another:
  - Different hardware ○ Different OS versions
- Virtual Machine approach provides a way of making code executable on any hardware – compiler produces code that is interpreted by the virtual machine
- Cross-platform compilation and code portability is another way, that compiles source code to multiple targets.

Distribution transparency:

To hide the separation of components. so that the system is perceived as a whole

- Access transparency (access to local or remote resources is identical相同)
- Location transparency (access without knowledge of location, If server moves and changes IP, DNS will always point to current location)
- Failure transparency (Tasks can be completed despite failures, by retransmission...)

Fault tolerance:

- Detecting: some types of failures can be detected, e.g. checksums can be used to detect corrupted data in a message, and some kinds of failure are hard to be certain about, e.g. the failure of a remote server
- Masking: some failures that have been detected can be hidden or made less severe, e.g. timeout and message retransmission
- Tolerating: it is sometimes impractical to try and handle every failure that occurs, sometimes it is better to tolerate them, e.g. failure is reported back to the user, e.g. web server not being available, try again later, or video is rendered with errors
- Recovery: failure sometimes leads to corrupted data and software can be designed so that it can recover the original state after failure, e.g. implementing a roll back mechanism.
- Redundancy: services can be made to tolerate failures using redundant components, e.g. multiple servers that provide the same service, as so called fail over.

Scalability: A system is considered to be scalable if it can handle the growth of the number of users.

- Scalability Challenges:
  - Cost of physical resources
  - Controlling the performance loss
  - Resources should not run out
  - Avoiding Performance bottlenecks

Concurrency:

- Multiple clients can access the same resource at the same time, in some cases for updates
- One approach to handling concurrency is making access sequential - slows down the system
- Semaphores supported by the operating system is a well accepted mechanism to handle concurrency

Openness: Openness refers to the ability of extend the system in different ways by adding hardware or software resources. Following are some approaches to address openness:

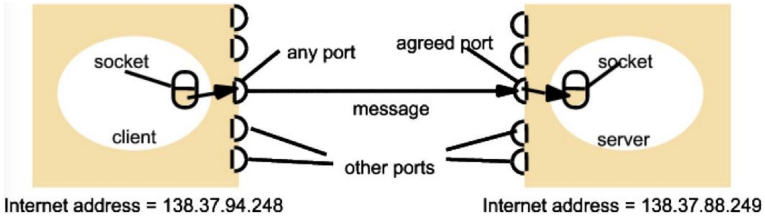
- Publishing key interfaces
- Allowing a uniform communication mechanism to communicate over the published interfaces
- Ensuring all implementations adhere to the published standards

Security: There has three aspects of security:

- Confidentiality (protection against disclosure to unauthorized individuals)
- Integrity (protection against alteration and corruption)
- Availability (protection against interference with the means of access)
- Security Mechanisms:
- Encryption (e.g. Blowfish, RSA)
- Authentication (e.g. passwords, public key authentication)
- Authorization (e.g. access control lists)

2_Inter-Process Communication: Network Programming using TCP Java Sockets 在Transport Layer

1. Briefly discuss three aspects of the Socket interface.



- Receiver process is bound to a local port.
- Socket can be used for sending and receiving.
- Each socket is associated with a protocol (UDP or TCP).

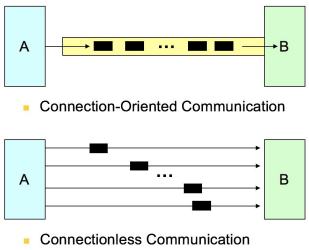
2. Briefly explain three possible failures that can happen when using UDP for communication.

UDP: User Datagram Protocol

- Provides a message passing abstraction.
- Is the simplest form of Interprocess Communication (IPC 进程间通信).
- Transmits a single message (called a datagram) to the receiving process.

Three possible failures:

- Data Corruption.
- Omission failures.
- Order.



3. Briefly explain three aspects of TCP that address issues not addressed by UDP.

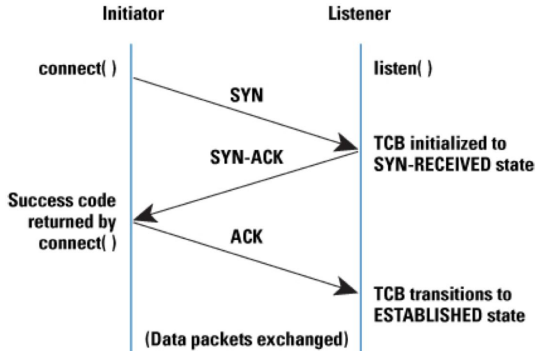
TCP: Transmission Control Protocol

- Provides an abstraction for a two-way stream.
- Streams do not have message boundaries. (比如用udp发”abc”,”def”,保持原样,但用tcp可能会变成”abcdef”...)
- Stream provide the basis for producer/consumer communication.
- Data sent by the producer are queued until the consumer is ready to receive them.
- The consumer must wait when no data is available.

Aspects:

- Message sizes: There is no limit on data size applications can use.
- Lost messages: TCP uses an acknowledgment scheme unlike UDP. If acknowledgments are not received the messages are retransmitted.
- Flow control: TCP protocol attempts to match the speed of the process that reads the message and writes to the stream.
- Message duplication or ordering: Message identifiers are associated with IP packets to enable the recipient to detect and reject duplicates and reorder messages in case messages arrive out of order.
- Message destinations: The communicating processes establish a connection before communicating. The connection involves a connect request from the client to the server followed by an accept request from the server to the client.

4. List the steps involved at the client and at the server to establish a TCP stream socket connection.



以下不属于workshop

5. JAVA

Java's .net package provides two classes: Socket – for implementing a client, ServerSocket – for implementing a server

(1) Implementing a Server (TCP)

1. Open the Server Socket:
```
ServerSocket server;
DataOutputStream os;
DataInputStream is;
server = new ServerSocket( PORT );
```
2. Wait for the Client Request:
```
Socket client = server.accept();
```
3. Create I/O streams for communicating to the client
```
is = new DataInputStream( client.getInputStream() );
os = new DataOutputStream( client.getOutputStream() );
```
4. Perform communication with client
```
Receive from client: String line = is.readLine();
Send to client: os.writeBytes("Hello\n");
```
5. Close sockets: `client.close();`

For multithreaded server:

```
while(true) {
  i. wait for client requests (step 2 above)
  ii. create a thread with "client" socket as parameter (the thread creates streams (as in step
  (3) and does communication as stated in (4). Remove thread once service is provided.
}
```

```
// SimpleServer.java: a simple server program
import java.net.*;
import java.io.*;
public class SimpleServer {
  public static void main(String args[]) throws IOException {
    // Register service on port 1234
    ServerSocket s = new ServerSocket(1234);
    Socket s1=s.accept(); // Wait and accept a connection
    // Get a communication stream associated with the socket
    OutputStream s1out = s1.getOutputStream();
    DataOutputStream dos = new DataOutputStream (s1out);
    // Send a string!
    dos.writeUTF("Hi there");
    // Close the connection, but not the server socket
    dos.close();
    s1out.close();
    s1.close();
  }
}
```

Server in Loop: Always up

```
// SimpleServerLoop.java: a simple server program that runs forever in a single thead
import java.net.*;
import java.io.*;
public class SimpleServerLoop {
  public static void main(String args[]) throws IOException {
    // Register service on port 1234
    ServerSocket s = new ServerSocket(1234);
    while(true)
    {
        Socket s1=s.accept(); // Wait and accept a connection
        // Get a communication stream associated with the socket
        OutputStream s1out = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream (s1out);
        // Send a string!
        dos.writeUTF("Hi there");
        // Close the connection, but not the server socket
        dos.close();
        s1out.close();
        s1.close();
    }
  }
}
```

(2) Implementing a Client (TCP)

1. Create a Socket Object:
```
client = new Socket( server, port_id );
```
2. Create I/O streams for communicating with the server.
```
is = new DataInputStream(client.getInputStream() );
os = new DataOutputStream( client.getOutputStream() );
```
3. Perform I/O or communication with the server:
   - Receive data from the server:
     `String line = is.readLine();`
   - Send data to the server:
     `os.writeBytes("Hello\n");`
4. Close the socket when done:
```
client.close();
```

```
// SimpleClient.java: a simple client program
import java.net.*;
import java.io.*;
public class SimpleClient {
  public static void main(String args[]) throws IOException {
    // Open your connection to a server, at port 1234
    Socket s1 = new Socket("jarrett.cis.unimelb.edu.au",1234);
    // Get an input file handle from the socket and read the input
    InputStream s1In = s1.getInputStream();
    DataInputStream dis = new DataInputStream(s1In);
    String st = new String (dis.readUTF());
    System.out.println(st);
    // When done, just close the connection and exit
    dis.close();
    s1In.close();
    s1.close();
  }
}
```

(3) Socket Exceptions

Server:

Client:

- public **ServerSocket**(int port) throws IOException
  - Creates a server socket on a specified port
  - A port of 0 creates a socket on any free port. You can use **getLocalPort**() to identify the (assigned) port on which this socket is listening
  - The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused
- Throws:
  - IOException - if an I/O error occurs when opening the socket
  - SecurityException - if a security manager exists and its checkListen method doesn't allow the operation

```
try {
    Socket client = new Socket(host, port);
    handleConnection(client);
}
catch(UnknownHostException uhe) {
    System.out.println("Unknown host: " + host);
    uhe.printStackTrace();
}
catch(IOException ioe) {
System.out.println("IOException: " + ioe);
    ioe.printStackTrace();
}
```

(4) UDP Server: repeatedly received a request and sends it back to the client

```
import java.net.*;
import java.io.*;
public class UDPServer{
    public static void main(String args[]){
    DatagramSocket aSocket = null;
      try{
        aSocket = new DatagramSocket(6789); // fixed port number
        byte[] buffer = new byte[1000];
        while(true){
          DatagramPacket request = new DatagramPacket(buffer, buffer.length);
          aSocket.receive(request);
          DatagramPacket reply = new DatagramPacket(request.getData(),
              request.getLength(), request.getAddress(), request.getPort());
          aSocket.send(reply);
        }
      }catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
       catch (IOException e) {System.out.println("IO: " + e.getMessage());}
    finally {if(aSocket != null) aSocket.close();}
  }
}
```

(5) UDP Client: Sends a Message and Gets reply

```
import java.net.*;
import java.io.*;
public class UDPClient
{
  public static void main(String args[]){
    // args give message contents and server hostname
    // "Usage: java UDPClient <message> <Host name> <Port number>"
    DatagramSocket aSocket = null;
     try {
        aSocket = new DatagramSocket();
        byte [] m = args[0].getBytes();
        InetAddress aHost = InetAddress.getByName(args[1]);
        int serverPort = 6789;  // Or Integer.valueOf(args[2]).intValue() if use <Port number> args[2]
        DatagramPacket request = new DatagramPacket(m,  args[0].length(), aHost, serverPort);
        aSocket.send(request);
        byte[] buffer = new byte[1000];
        DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
        aSocket.receive(reply);
        System.out.println("Reply: " + new String(reply.getData()));
     }
     catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
     catch (IOException e){System.out.println("IO: " + e.getMessage());}
     finally
    {
       if(aSocket != null) aSocket.close();
    }
  }
}
```

3_Multithreaded Programming using Java Threads

1. When do we need multi-threading?

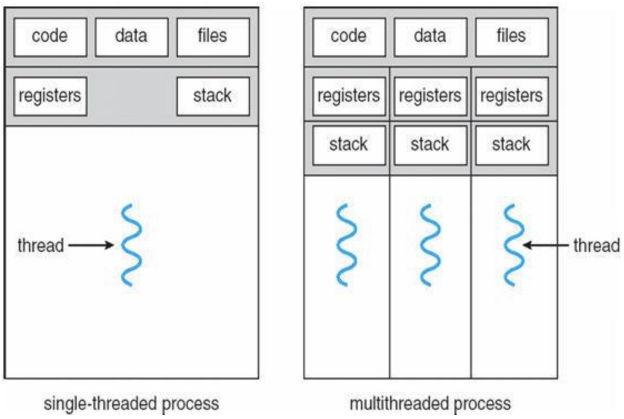
- Allow multiple operations performed at the same time to achieve concurrency at the application level
- Improve application performance

- Improve resource utilization rate
- Application scenarios
    - Implementation of reactive user interfaces.
    - Handling concurrent requests and serving multiple clients
    - Non-blocking I/O operations.
    - Asynchronous behaviour.
    - Timer and alarms implementation.

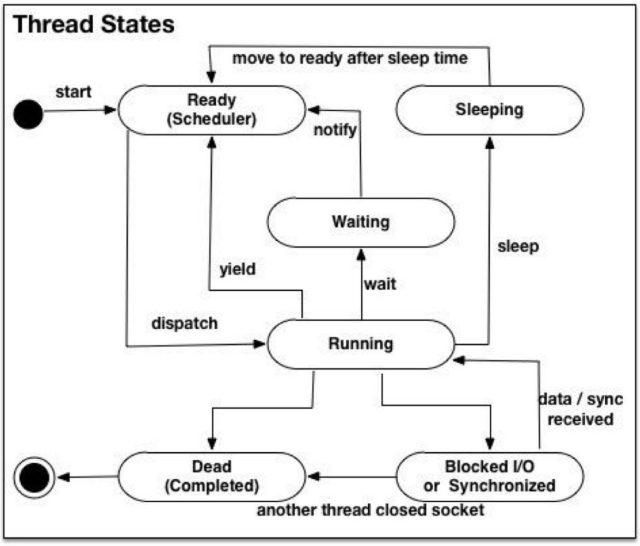
2. What is the relationship between thread and process?

Threads are light-weight processes within a process

- Threads are objects of process in Java consisting of a statically ordered sequence of instructions.
- Threads share code, data, files, and I/O in heap memory
- Threads have their independent stack memory and registers



3. How do threads run on CPUs?



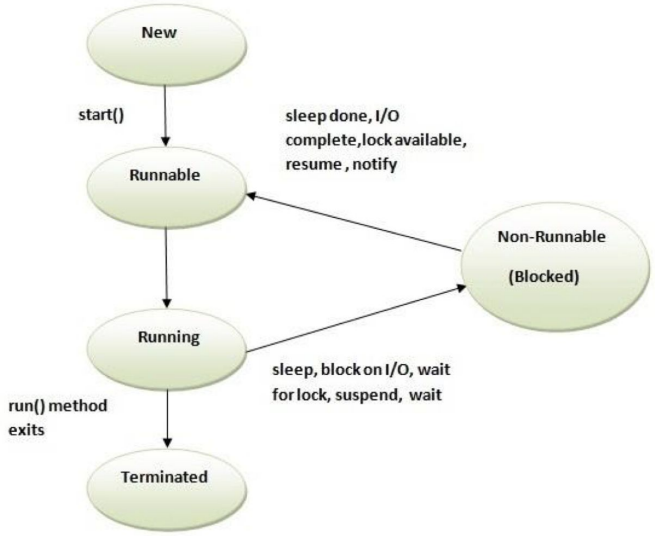
- Threads have the illusion of their own CPU
    - But in reality, they have to share one (or more) CPUs

- ○ This is handled by the OS with its thread scheduling module (time-sharing)
- ○ The OS can run or suspend a thread at any time

● Dispatch ○ Thread gets the CPU resources to run
● Yield ○ Thread gives hint to the thread scheduler that it is ready to pause its execution.
● Sleep ○ Sleep for a period
● Wait ○ Causes current thread to wait until another thread invokes the notify() method for this object or notifyAll().
● Notify ○ To wake a thread up

4. What is a thread and life cycle of a thread?

- A Thread is a piece of code that runs in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are used to express concurrency on both single and multiprocessors machines.
- Threads share the same address space and therefore can share both data and code



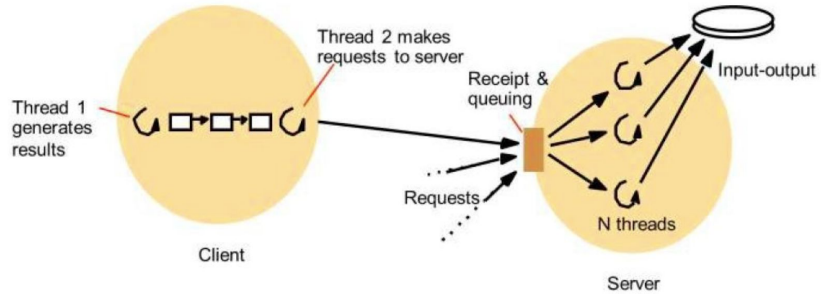
5. What is synchronous access to shared resources and how can we achieve it?

- If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to the data.
- Use “synchronized” method:

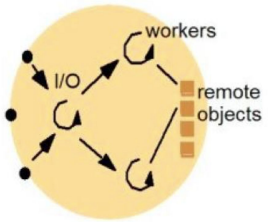
```
public synchronized void update() {
        ...
}
```

6. Compare the worker pool multi-threading architecture with the thread-per-request architecture. (Architecture for Multi-thread Servers)

Worker pool architecture



The server creates a fixed number of threads called a worker pool. As requests arrive at the server, they are put into a queue by the I/O thread and from there assigned to the next available worker thread.
Server creates worker pool → request comes in, put into a queue → assigned to an available worker thread. Server down时pool destroy



a. Thread-per-request

Thread created for each request, when the request is finished, the thread is deallocated重分配？

7. What is deadlock and how to avoid it?

● Deadlock can occur when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread
● Deadlock can be easily avoided by resource ordering.
● With this technique, assign an order on all the objects whose locks must be acquired and ensure that the locks are acquired in that order.
● Example:
  ○ Thread 1: lock A lock B
  ○ Thread 2: wait for A lock C (when A is locked)
  ○ Thread 3: wait for A wait for B wait for C
● An example application to illustrate wait and notify
  ○ Thinking of an implementation of blocking queue that has a fixed size
  ○ The queue needs to be accessed by multiple threads i.e. thread-safe
    Remember, two unsynchronized threads accessing the queue at the same time may cause conflict.
● What methods are needed for a queue abstraction?

```java
public class BlockingQueue<T> {

    private Queue<T> queue = new LinkedList<T>();
    private int capacity;

    public BlockingQueue(int capacity) {
        this.capacity = capacity;
    }

    public synchronized void put(T element) throws InterruptedException {
        while(queue.size() == capacity) {
            wait();
        }

        queue.add(element);
        notify(); // notifyAll() for multiple producer/consumer threads
    }

    public synchronized T take() throws InterruptedException {
        while(queue.isEmpty()) {
            wait();
        }

        T item = queue.remove();
        notify(); // notifyAll() for multiple producer/consumer threads
        return item;
    }
}
```

以下非Workshop:

8. Middleware: Middleware is a layer of software (system) between Applications and Operating System (OS) powering the nodes of a distributed system.

Network Operating System (NOS): A combination of middleware and NOSs provides an acceptable balance between the requirement of autonomy and network transparency.

10. JAVA Thread

(1) Extending Thread class

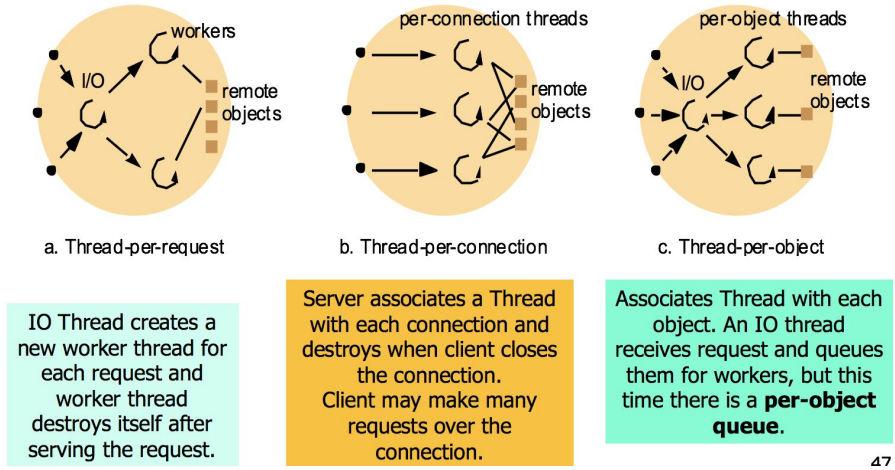
```java
class MyThread extends Thread {
        public void run() {
                System.out.println(" this thread is running ... ");
        }
}

class ThreadEx1 {
        public static void main(String [] args  ) {
           MyThread t = new MyThread();
            t.start();
        }
}
```

(2) Threads by implementing Runnable interface

```java
class MyThread implements Runnable  {
        public void run() {
                System.out.println(" this thread is running ... ");
        }
}

class ThreadEx2 {
        public static void main(String [] args  ) {
                Thread t = new Thread(new MyThread());
                 t.start();
        }
}
```

```java
A threadA=new A();
B threadB=new B();
C threadC=new C();
threadC.setPriority(Thread.MAX_PRIORITY);
threadB.setPriority(threadA.getPriority()+1);
threadA.setPriority(Thread.MIN_PRIORITY);
```

11. Alternative server threading architectures



worker-pool: 不用每次create, 不用go through whole lifecycle, 节省时间
其他: 不用管server是否busy, pool是否满, 可以create any number of pool
per request: take more time to serve
per connection: 适合 want to make lots of request

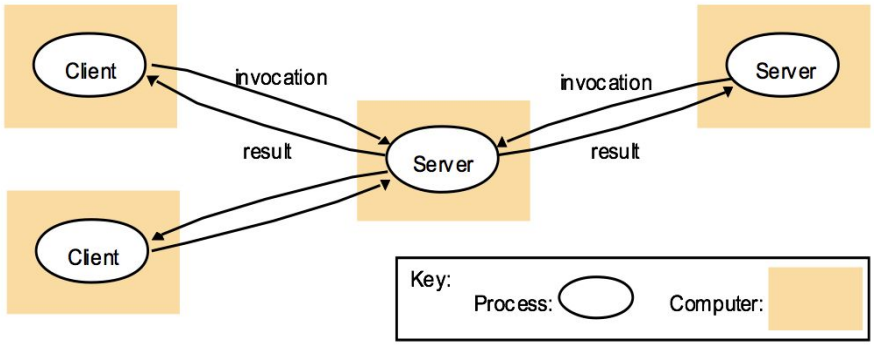
```java
package Server;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

import javax.net.ServerSocketFactory;

public class Server {

    // Declare the port number
    private static int port = 3000;

    // Identifies the user number connected
    private static int counter = 0;

    public static void main(String[] args) {
        ServerSocketFactory factory = ServerSocketFactory.getDefault();
        try(ServerSocket server = factory.createServerSocket(port)){
            System.out.println("Waiting for client connection..");

            // Wait for connections.
            while(true){
                Socket client = server.accept();
                counter++;
                System.out.println("Client "+counter+": Applying for connection!");


                // Start a new thread for a connection
                Thread t = new Thread(() -> serveClient(client));
                t.start();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }


    }

    private static void serveClient(Socket client){
        try(Socket clientSocket = client){
            // Input stream
            DataInputStream input = new DataInputStream(clientSocket.
                    getInputStream());
            // Output Stream
            DataOutputStream output = new DataOutputStream(clientSocket.
                    getOutputStream());
            System.out.println("CLIENT: "+input.readUTF());
            output.writeUTF("Server: Hi Client "+counter+" !!!");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

```java
package Client;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;

public class Client {

    // IP and port
    private static String ip = "localhost";
    private static int port = 3000;

    public static void main(String[] args) {
        try(Socket socket = new Socket(ip, port);){
            // Output and Input Stream
            DataInputStream input = new DataInputStream(socket.
                    getInputStream());
            DataOutputStream output = new DataOutputStream(socket.
                    getOutputStream());

            output.writeUTF("I want to connect!");
            output.flush();

            while(true){
                String message = input.readUTF();
                System.out.println(message);

            }

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {

        }

    }

}
```

4_Distributed System Models

1. Briefly explain the difference between a client-server architecture and a peer- to-peer architecture.

(1) Client-Server



(Querying a web server, which could then query a mysql or oracle database before returning the content of a page Web server is a client of the database server; A search engine, as well as serving search requests from clients, crawls other websites to keep its information current; Search engine is both a server and a client for other web servers.)

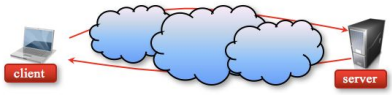
Client processes interact with individual server processes in a separate computer in order to access data or resource. The server in turn may use services of other servers.

Example:

A Web Server is often a client of file server.

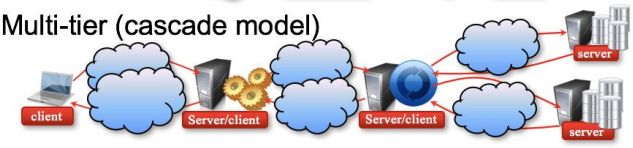
Browser -> search engine -> crawlers -> other web servers.

- Two-tier model (classic)

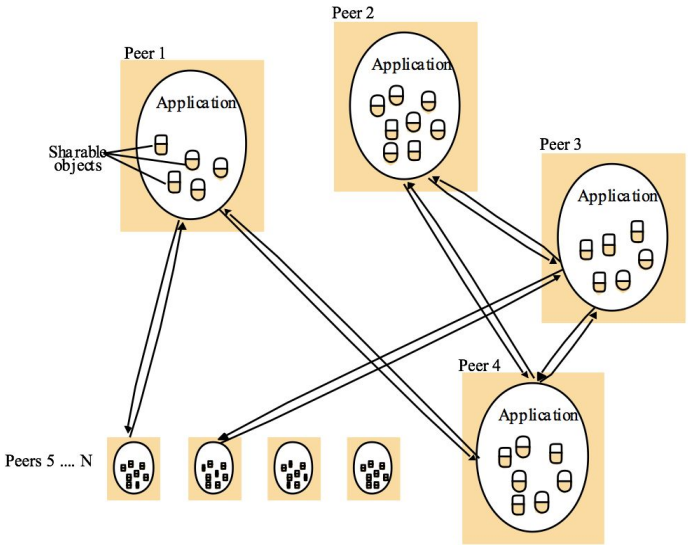


- Three-tier (when the server, becomes a client)

- Multi-tier (cascade model)



(2) Peer-to-Peer



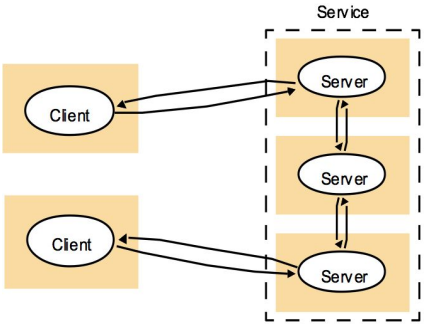
All of the processes play similar roles, interacting cooperatively as peers to perform distributed activities or computations without distinction between clients and servers. E.g., music sharing systems Gnutella, Napster, Kaza, etc.

Distributed “white board” – users on several computers to view and interactively modify a picture between them.

- No central point of failure (reliable)
- No central point of control (difficult to deny service for adversaries)
- Some peers will typically contribute more than others (I.e. seed or super-peer)

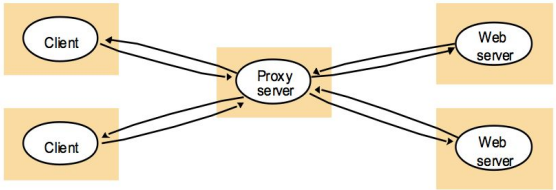
2. Briefly explain each of the following distributed system architecture variations, giving also a reason or a benefit for its use:

- Services provided by multiple servers

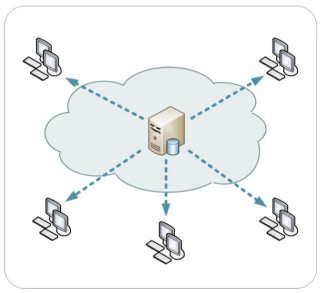


- Services may be implemented as several server processes in separate host computers (redirected (via DNS and other means) to one of them) interacting as necessary to provide a service to client processes
- Servers may
    - Partition the set of objects on which the service is based and distribute those objects between themselves
    - Maintain replicated copies of them on several hosts
- Improve performance and reliability

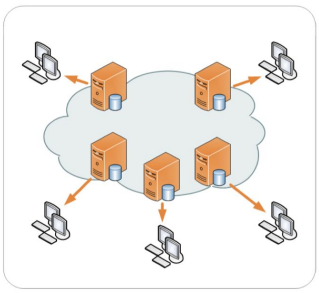
- Proxy代理 servers and caches



- Cache
    - A store of recently used data objects that is closer to the client
    - They may be co-located with each client or they may be located in a proxy server (at ISP level and at edge/gateway levels) that can be shared by several clients
- Increase the availability and performance of the service by reducing the load on the wide area network and web servers
- Proxy servers can take on other roles --- better reliability
- Improved security
- Access restriction
- Privacy protection



Single server distribution



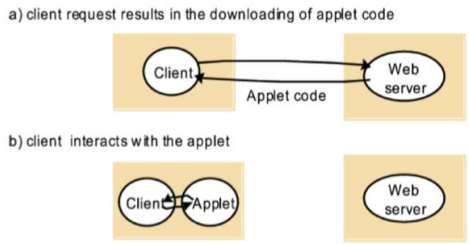
CDN scheme of distribution

Caching can even be used for dynamic data (such as a google search). This reduces the load on the web servers and improves the performance for end users by reducing the time taken for a dynamic request.

- Mobile code and Mobile Agents

Similar to client-server model but client takes on more responsibility

- Mobile code
  - Mobile Code is downloaded to the client and is executed on the client (e.g. applet).
  - Good interactive response (As code is executed locally)
  - Security threat (cannot control the client environment, e.g. client applet not suitable for online banking!)



- Mobile agents
  - Mobile agents are running programs that includes both code and data that travels from one computer to another.
  - They process data at the data source, rather than fetching(取) it remotely

    Less communication overhead by replacing remote invocations with local ones
  - Security threat (Agents themselves can be vulnerable – they may not be able to complete task if they are refused access, 并且Potential security threat to the resources in computers they visit)

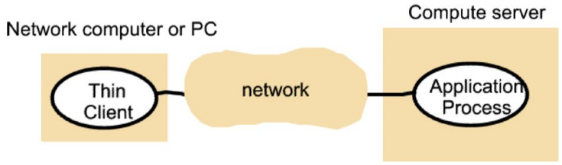
Can be a useful approach (e.g. data indexing/mining) if data is spread over wide geographical areas. Bring application (which is small) to data rather than the other way around.

- Network computers

- Download their operating system and application software from a remote file system. Applications are run locally.

- Thin clients easy to maintain and secure

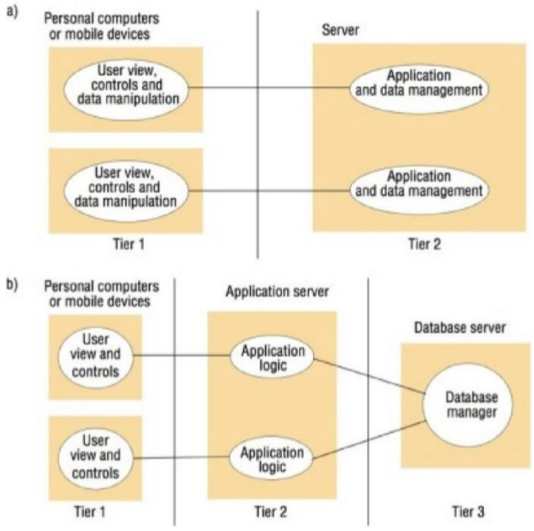
- Move complexity away from the end-user device
- Local user interface, remote services or applications
- Few assumptions or demands on the client device



- Tiered分层 Architecture

- Tiered architectures are complementary补充 to layering, which deals with horizontal organization of services.

- Layering deals with vertical organization of services



3. Briefly explain the purpose of the following fundamental models and explain two important considerations for each:

(1) Interaction Model.

Models the interaction between processes of a distributed system

- e.g. interaction between clients and servers or peers.

Communication can be discrete messages or packets, or constant streams/broadcasts (e.g. satellite or sensor data).

- Performance of communication channels.
  - Latency ○ Bandwidth ○ Jitter
- Computer clocks and timing events
  - Why the timestamps between two processes can vary?
    - Initial time setting being different
    - Differences in clock drift漂移 rates

Two variants of the interaction model 交互模型的两个变体

Synchronous DS – hard to achieve: 对进程每一步的执行时间有一个上限和下限，传递的信息需在一个已知的时间范围内收到，每个进程携带一个本地时钟并且与实际时间偏移率在一直范围内（需要保证足够的处理器周期和网络能力以及准时的时钟）

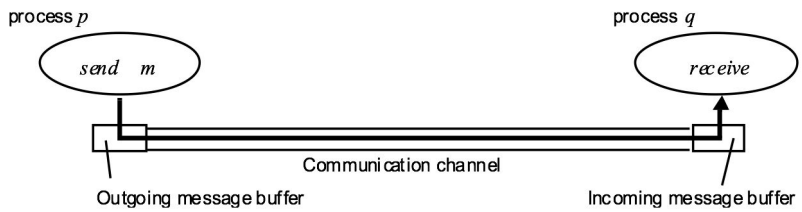
Asynchronous DS: There is NO bounds on: Process execution speeds, Message transmission delays, Clock drift rates. 没有任何假设

Interaction Model: Event Ordering事件排序 理想化知道准确时间，实际不能，可推逻辑顺序(回信比收信晚)

(2) Failure Model.

Classifies the failures of processes and communication channels in a distributed system

- Omission failures
  - Omission failures refers to cases where a process or a communication channel fails to perform what is expected to do



Message is not accepted from process 's outgoing buffer onto the link

Message is accepted but does not reach q's incoming buffer (e.g. router failure or overload, R.E.D, or link down)

- Arbitrary failures
  - Refers to any type of failure that can occur in a system

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

- Timing failures
  - These failures occur when time limits set on process execution time, message delivery time and clock rate drift.
  - More relevant to synchronous systems

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

- Masking Failures

It is possible to construct reliable services from components that exhibit failures.

For example, multiple servers that hold replicas of data can continue to provide a service when one of them crashes (use TCP).

A knowledge of failure characteristics of a component can enable a new service to be designed to mask the failure of the components on which it depends:

Checksums are used to mask corrupted messages.

(3) Security Model.

Identifies the possible threats to **processes** and **communication channels**, as well as protecting encapsulated objects against unauthorized access.

- Encryption. (Most approaches use a public/private key pair system (e.g. SSH, HTTPS/SSL, VPN) where traffic is encrypted by the sender with their private key and decrypted and verified by the receiver using the sender's public key)
- Authentication. (The server should verify the identity of the user behind each operation and checking that they have sufficient access rights to perform the requested operation on the particular object, rejecting those who do not.)

- Secure Channel. (Encryption and authentication are use to build secure channels)

Security Model - How to Perform Encryption

- Secret Key.
- Public/Private key pair.
- Establishing a secure channel (SSL).

4. Explain the difference between a synchronous protocol and an asynchronous protocol.

- Synchronous communication blocks on both send and receive operations.
  - When a send is issued the sending process is blocked until the receive is issued.
  - Whenever the receive is issued the process blocks until a message arrives.
- In Asynchronous communication the send is nonblocking.
  - The sending process returns as soon as the message is copied to a local buffer and the transmission of the message proceeds in parallel.
  - Receive operation can be blocking or non-blocking (non-blocking receives are not normally supported in today's systems).

以下非workshop

5. DS model的三种分类方式:

(1) Physical Model: 根据计算机及其互联网方面硬件分类

① Early DSs [70-80s]: LAN-based, 10-100 nodes, 支持少量服务(如共享打印机和文件传输), 单个系统大部分同构, 开放性不是主要问题

② Internet-scale DSs [early 90-2005]: Clusters, grids, P2P (with autonomous nodes)

③ dynamic nodes in mobile systems that offer location-aware services, Clouds with resource pools offering services on pay-as-you-go basis. 节点可移动位置, 规模超大

(2) Architectural model: the placement of its parts and relationship between them. 独立指定组件以及这些组件间的关系, 例如Client-Server (CS) and peer process models

(3) Fundamental Models: a formal description of the properties性能 that are common in all of the architectural models 描述大多数分布式系统面临的单个问题的解决方案

6. Middleware

A layer of software whose purpose is to mask heterogeneity present in distributed systems and to provide a convenient programming model to application developers.

Major Examples: Sun RPC (Remote Procedure Call), Sun Java RMI (Remote Method Invocation)

Modern Middleware Examples: Microsoft .NET, Google AppEngine, Microsoft Azure

5_Operating System Architecture and Distributed Systems

1. Discuss the difference between a network operating system and a distributed operating system.

- A networked operating system provides support for networking operations. The users are generally expected to make intelligent **use of the network commands and operations** that are provided. Each host remains autonomous in the sense that it can continue to operate when disconnected from the networking environment. 无法连接到网络也能操作
- A distributed operating system tries to abstract the network from the user and thereby **remove** the need for the user to specify how the **networking commands and operations** should be undertaken. This is sometimes referred to as providing a single system image. Each host may not have everything that would be required to operate on its own, when disconnected from the network. 当无法连接到网络时不能完全操作

Network operating system (例子: UNIX, Window)

- Users retain保持 autonomy独立 in managing their own processing resources本机资源
- It does not manage processes across the nodes
- Provides support for networking operations

运行网络操作系统的节点能独立管理自己的进程资源, 有多个系统映像(System Image), 每个节点一个, 用户能ssh远程登录到另一节点运行那里的进程, 但不能远程管理进程

Distributed operating system

- Users are never concerned with where their programs run, or the location of any resources
- Has control over all the nodes in the system, and it transparently locates new processes at whatever node suits its scheduling policies
- Each host may not have everything that would be required to operate on its own
- Single system image
- Not widely adopted

有单一的系统映像, 任一节点都能控制系统中的所有节点

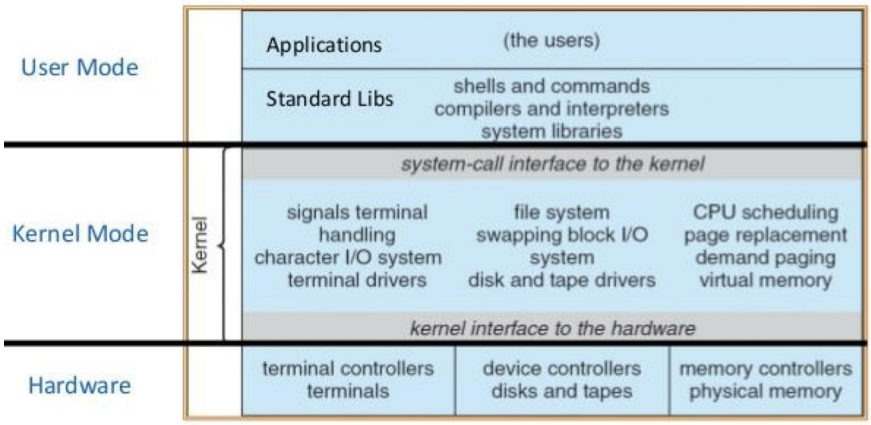
没有普遍应用的原因是用户已适应现有的Network OS, 另一原因是即使在小公司, 用户也倾向于独立管理自己的机器

2. What are the core OS components?

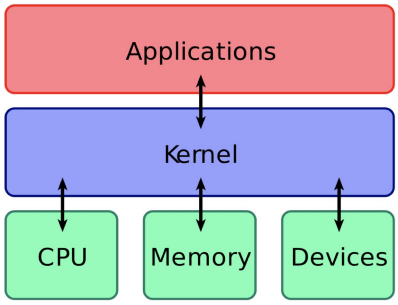
- **Process manager** -- Handles the creation of processes, which is a unit of resource management, encapsulating the basic resources of memory (address space) and processor time (threads).
- **Thread manager** -- Handles the creation, synchronization and scheduling of one or more threads for each process. Threads can be scheduled to receive processor time.
- **Communication manager** -- Handles interprocess communication, i.e. between threads from different processes. In some cases this can be across different hosts
- **Memory manager** -- Handles the allocation and access to physical and virtual memory. Provides translation from virtual to physical memory and handles paging of memory.

- **Supervisor** -- Handles privileged特权 operations, i.e. those that directly affect shared resources on the host, e.g. to and from an I/O device. The supervisor is responsible for ensuring that host continues to provide proper service to each client.

A Real-World Example - UNIX (Linux和Mac是其衍生版本)

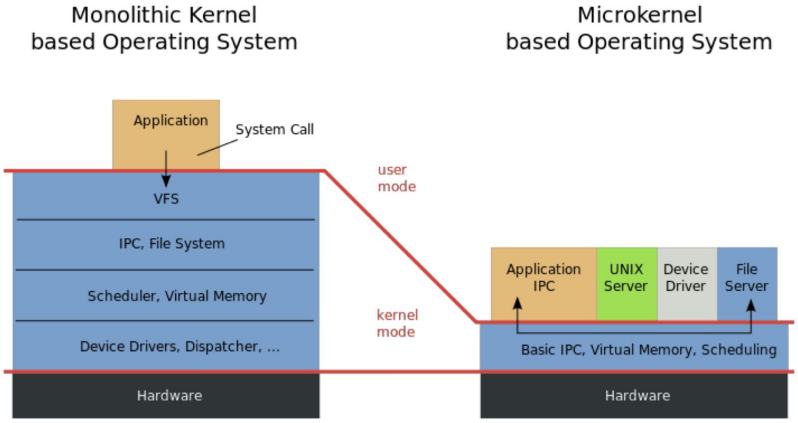


3. What is the kernel?



- Part of the Operating System
- Has full access to the host's resources
- Kernel begins execution after the host is powered up and continues to execute while the host is operational
- The kernel has access to all resources and shares access to all other processes that executing on the host

Comparison of Kernel Implementations



4. What is Supervisor Mode vs User Mode?

- Operating modes supported by the hardware at the machine instruction level.
- Supervisor / Kernel mode -- instructions that execute while the processor is in supervisor (or privileged) mode are capable of accessing and controlling every resource on the host.
- User mode -- instructions that execute while the processor is in user (or unprivileged) mode are restricted, by the processor, to only those accesses defined or granted by the kernel.

5. Discuss the potential advantages and disadvantages of using of multiple threads compared to multiple processes.

Advantages

• Cheaper to allocate/deallocate
• Faster switching
• Easy to share resources via shared memory.

Disadvantages

• Synchronization overhead of shared data
• Shared process memory space
• Program debugging (synchronization, non-deterministic timing, accidental data corruption)

不属于workshop:

6. A open DS 的要求

(1) 每个组件上只有必须的system software, 比如client端loading redundant modules是浪费
(2) Allow the software implementing any particular service to be changed independent of other facilities. 比如加一个鼠标或其他device, 会有对应的驱动, remove就不需要这个驱动, 可以dynamic增加或remove任一设备, 不会影响其他设备
(3) Allow for alternatives of the same service to be provided, when this is required to suit different users or applications. 要allow同一service的替代品
(4) Introduce new services without harming the integrity of existing ones.新services不会影响现存的

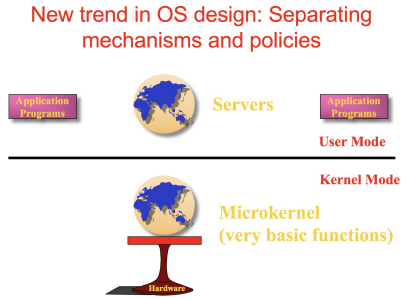
7. Separating Mechanisms and Policies in OS and DS 分离机制和策略

mechanisms (those parts of a system implementation that control the authorization of operations and the allocation of resources 例如授权给一个操作或者分配一个资源的实现) should not dictate指挥 (or overly restrict过度限制) the policies according to which decisions are made about which operations to authorize, and which resources to allocate 决定改给哪个操作授权或是分配哪个资源, 这些都是vary from application to application and service to service的.
即mechanisms只管基础操作, policies对于不同情况采取不同策略

For example, an ideal scheduling system一个理想的调度系统 would provide **mechanisms** that enable a multimedia application such as videoconferencing to meet its real-time demands while coexisting with a non-real-time application such as web browsing 实时的视频会议和不需要实时的浏览器都可以用相同的mechanism

Kernel would provide only the most basic **mechanisms** upon which the general resource management tasks. 即Kernel负责提供mechanisms

Server modules would be dynamically loaded as required, to implement the required RM policies for the currently running applications. 即Server modules负责提供Policies



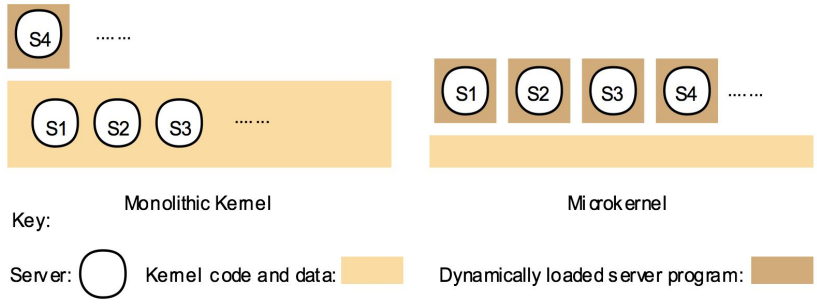
8. Two key examples of kernel design approaches

(1) Monolithic

(2) Microkernel

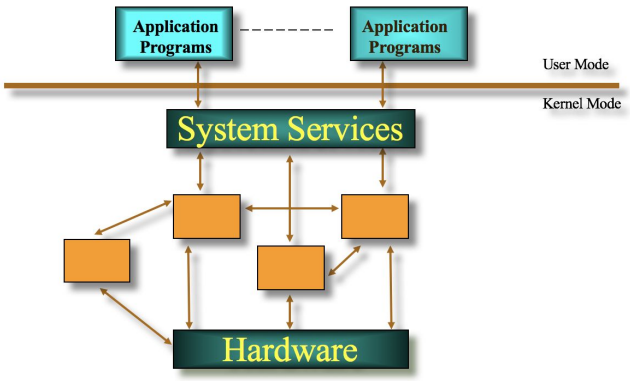
These two designs differ primarily in the decision as to what functionality belongs in the kernel and what is left to server processes that can be dynamically loaded to run on top of it.

这两种不同点在于哪些functionality属于kernel哪些留给server processes动态加载



9.Predominantly 3 types of OS

(1) Monolithic OS



Better application Performance

Difficult to extend

Ex: MS-DOS

Disadvantages of Monolithic OS:

<u>It is massive:</u>

It performs all basic OS functions and takes up in the order of megabytes of code and data

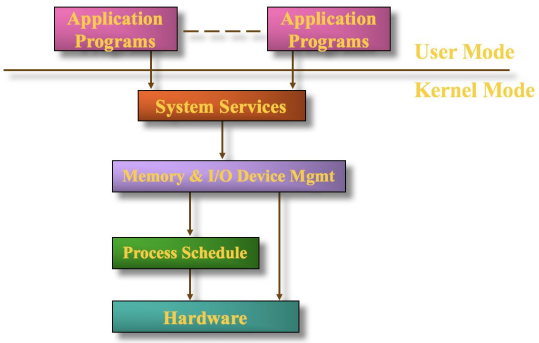
<u>It is undifferentiated:</u>

It is coded in a non-modular way (traditionally) although modern ones are much more layered.

<u>It is intractable:</u>

Altering any individual software component to adapt it to changing requirements is difficult.

(2) Layered OS



Easier to enhance

Each layer of code access lower level interface

Low-application performance

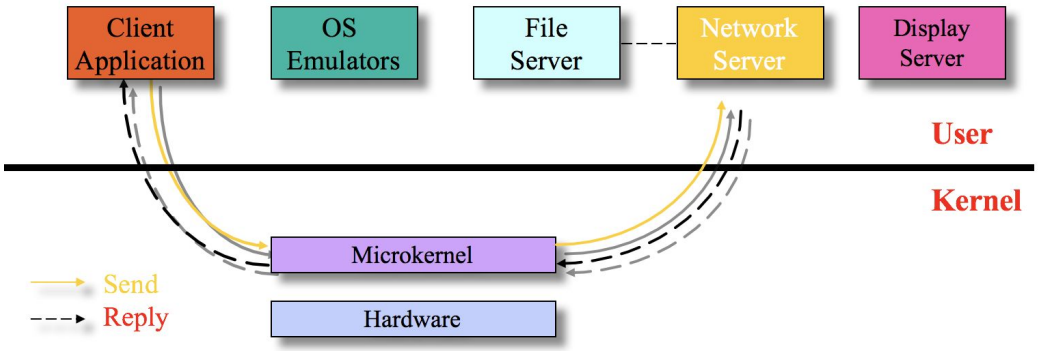
Ex : UNIX

(3) Microkernel-based OS

Compared to monolithic, **<u>microkernel design provides only the most basic abstractions</u>**分离机制和策略 (address space, threads and local IPC)

All other system services are provided by servers that are **dynamically loaded precisely on those computers** in the DS that **require them**.

Clients access these system services using the kernel's message-based invocation mechanisms.



Tiny OS kernel providing basic primitive (process, memory, IPC)

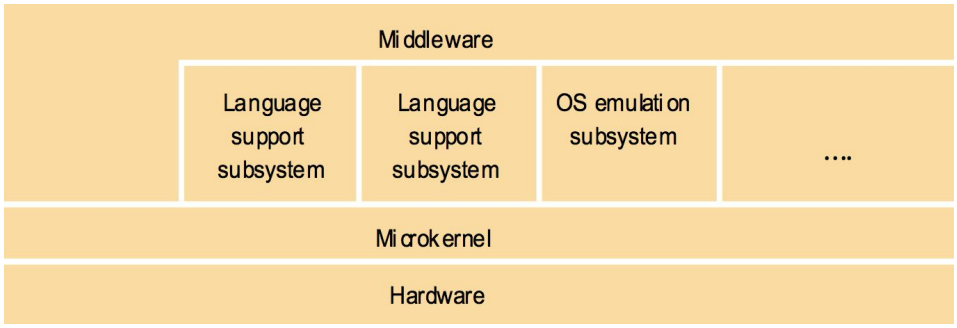
Traditional services becomes subsystems

OS = Microkernel + User Subsystems

Ex: Mach, QNX, Windows NT!

The role of the microkernel (MK)

MK appears as a layer between H/W and a layer of major system components (subsystems). If performance, rather than portability is goal, then middleware may use facilities of MK directly.



10. Comparison: Monolithic and Micro-kernel OS Design

The main advantages of a MK-based OS:

A relative small kernel is more likely to be free of bugs than one that is larger and complex.
Extensibility and its ability to enforce modularity behind memory protection boundaries

The advantage of a monolithic OS:

Relative efficiency with which operations can be invoked is high because even invocation to a separate user-level address space on the same node is more costly.

11. Hybrid Approaches

Many modern OSs follow hybrid approach in OS structure. E.g., Windows NT.

Pure microkernel OSs such as Chorus & Mach have changed over time to allow servers to be loaded dynamically into the kernel address space or into a user-level address space.

Some OSs (such as SPIN) use event-based model as a mechanism for interaction between modules grafted into the kernel address space.

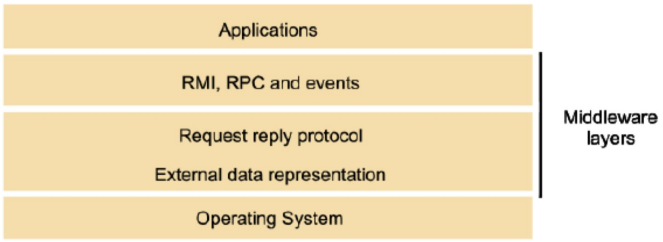
12. OSs provide various types of facilities/services to support middleware for distributed system:

encapsulation, protection, and concurrent access and management of node resources.

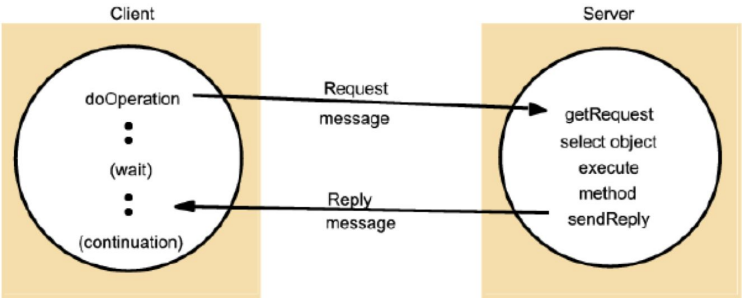
6_Distributed Objects Programming - Remote Invocation

1. Remote Invocation

- **A set of information exchange protocols at the Middleware layer**



- **Define and explain the procedure of the Request-Reply protocol**



Exchange protocol for the implementation of remote invocation in a distributed system. We discuss the protocol based on three abstract operations: doOperation, getRequest and sendReply

- `public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)`
  - Sends a request message to the remote server and returns the reply
  - The arguments specify the remote server, the operation to be invoked and the arguments of that operation
- `public byte[] getRequest ()`
  - Acquires a client request via the server port
- `public void sendReply (byte[] reply, InetAddress clientHost, int clientPort)`
  - Sends the reply message reply to the client at its Internet address and port

- **Possible Issues in Request-Reply**
  - Request timeouts
    - Retry request message after timeout
    - Do nothing
  - Reply timeouts
    - Retransmit results after timeout
    - Do nothing
  - Receiving duplicate requests/replies
    - Discard duplicate messages
    - Perform the same action again if the logic is idempotent幂等
  - History - if servers have to send replies without re-execution, a history has to be maintained
- **Invocation Semantics** 语义
  - Different issue handling strategies lead to different invocation semantics
  - Invocation semantics
    - Define what the client can assume about the execution of the remote procedure
    - Offer different reliability guarantees in terms of the number of times that the remote procedure is executed
  - Classification of **Request/Reply protocols** based on invocation semantics **(Exchange Protocols)**
    - The request protocol (R)
    - The request-reply protocol (RR)
    - The request-reply-acknowledge protocol (RRA)

Name	Messages sent by		
	Client	Server	Client
R	Request		
RR	Request	Reply	
RRA	Request	Reply	Acknowledge reply

- Three main design decisions related to implementations of the request/reply protocols are:
  - Strategy to retry request message
  - Mechanism to filter duplicates
  - Strategy for results retransmission

- **Different Invocation Semantics**

  - Maybe:

  - The remote procedure call may be executed once or not at all.
  - Unless the caller receives a result, it is unknown as to whether the remote procedure was called.
  - Suitable for applications in which occasional failed calls are acceptable

  - At-least-once:

  - Either the remote procedure was executed at least once, and the caller received a response, or
  - The caller received an exception to indicate the remote procedure was not executed at all.
  - Suitable for idempotent幂等 operations.

  - At-most-once:

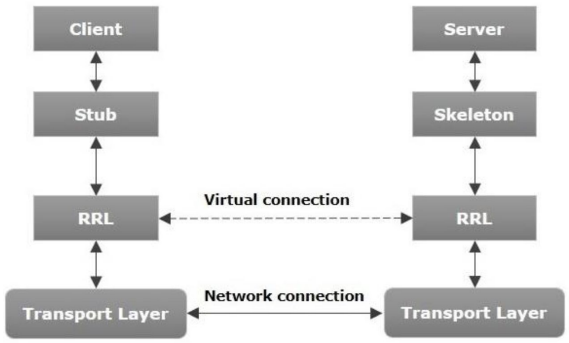
  - The remote procedure call was either executed exactly once, in which case the caller received a response, or
  - It was not executed at all and the caller receives an exception.
  - Suitable for non-idempotent operations.

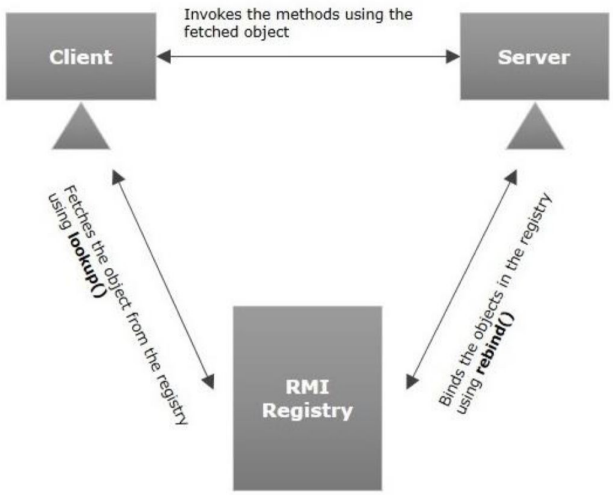
Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

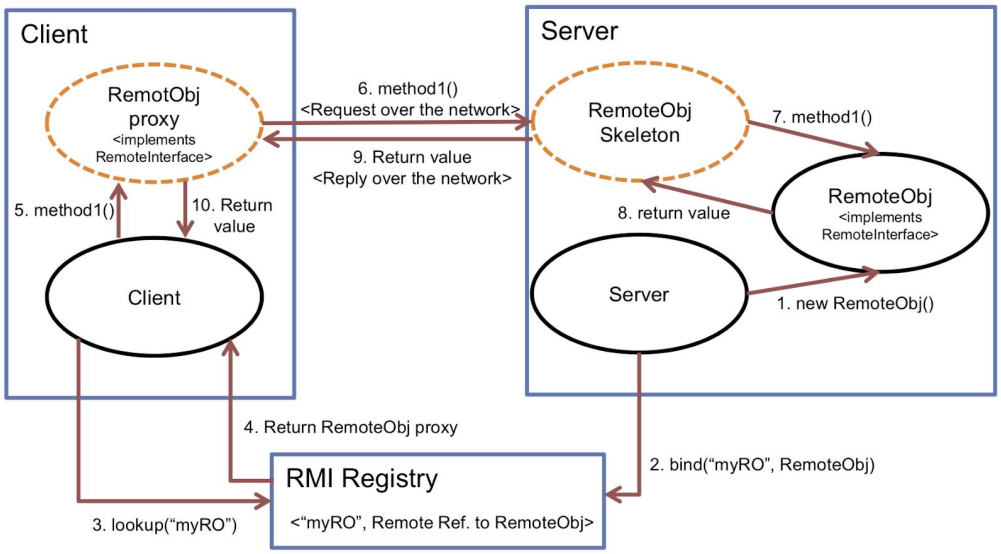
后两种区别:重新执行和重传结果

Java RMI supports at-most-once invocation. Sun RPC supports at-least-once semantics.

2. Explain the steps involved in Java RMI to build a distributed system.







- Define the remote interface
  - Defines the methods that can be remotely invoked
  - Extends java.rmi.Remote interface
  - All methods throw java.rmi.RemoteException
  - Needs to be included in the client and in the server programs

- Server program
    - Create the remote object
    - Implements the remote interface
    - Extends UnicastRemoteObject
    - Implement the actual server
        - Export the remote object into the Java RMI runtime so that it can receive remote calls
        - Locate the RMIregistry and publish the remote object on it (the registry stores <name, remoteObjRef> pairs.
- Client program
    - Locate the RMIregistry and lookup the remote object by name
    - Invoke methods on the remote object as if it was a local one

以下非Workshop:

3. Two high level programming models

(1) **Remote Procedure Call** - an extension of the conventional procedure call model

(2) **Remote Method Invocation** - an extension of the object-oriented programming model.

RMI是面向对象的JAVA RPC, RPC不能传对象

4. Typical Message Content in a request-reply protocol

A message in a request-reply protocol typically contains a number of fields as shown below:

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*array of bytes*

5. Failure Model

If operations are implemented using UDP, then they suffer from the same communication failures: Lost messages, Out of order.

In addition, the protocol can suffer from the failure of processes (crash failures).

6. Client-Server Communication

- Client-server communication normally uses the synchronous request-reply communication paradigm
- Involves send and receive operations
- TCP or UDP can be used - TCP involves additional overheads:

    Redundant acknowledgements

    Needs additional messages for establishing connection

    Flow control is not needed since the number of arguments and results are limited
- Longevity寿命 of TCP connections is an important design issue

7. Distributed Object Concepts

- **Remote Object**

An object that can receive remote invocations
Can also receive local invocations
Can invoke methods in local objects as well as other remote objects

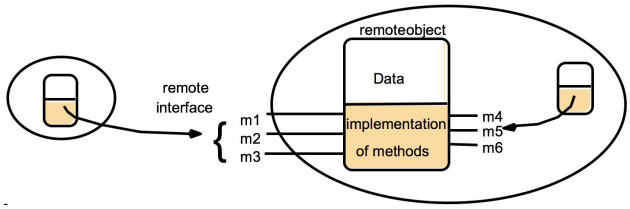
- **Remote Object Reference**

An identifier识别码 that can be used throughout a distributed system to refer to a particular unique remote object.
Other objects can invoke the methods of a remote object if they have access to its remote object reference.

32 bits	*32 bits*	*32 bits*	*32 bits*	
Internet address	port number	time	object number	interface of remote object

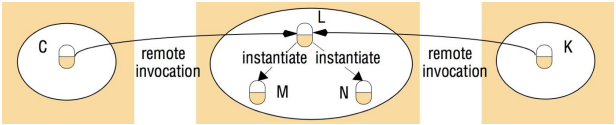
- **Remote Interface**

Defines the methods that can be invoked by external processes.
Remote objects implement the remote interface.

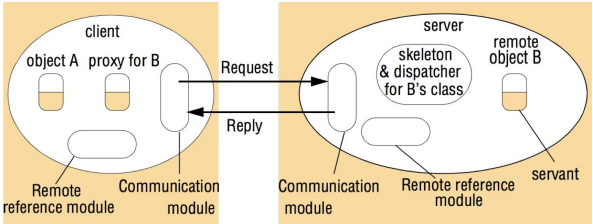


- **Actions**

Actions can be performed on remote objects (objects in other processes)
An action could be executing a remote method defined in the remote interface or creating a new object in the target process
Actions are invoked using Remote Method Invocation (RMI)



8. Communication Module and Remote Reference Module



communication module和remote reference module都是client端和server端都有

The **server module** queries the **remote reference module** to obtain the local reference of the object and passes the local reference to the dispatcher调度员 for the class.
Remote Reference Module has a **remote object table** that records the correspondence between local object references in that process and remote object references (system-wide)
The remote object table contains an entry for each:

Remote object reference held by the process

Local proxy

Entries are added to the remote object table when:

A remote object reference is passed for the first time

When a remote object reference is received and an entry is not present in the table

9. RMI Software

● **Software layer** that lies between the application and the communication and object reference modules

● **Proxy**: Plays the role of a local object to the invoking object. There is a proxy for each remote object which is responsible for:

Marshalling编组 the reference of the target object, its own method id and the arguments and forwarding them to the communication module

Unmarshalling解组 the results and forwarding them to the invoking object

● **Dispatcher**: There is one dispatcher for each remote object class. Is responsible for mapping to an appropriate method in the skeleton based on the method ID

● **Skeleton**: Is responsible for:

Unmarshalling the arguments in the request and forwarding them to the servant

Marshalling the results from the servant to be returned to the client

10. Java RMI Example

```java
package remote;
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * RMI Remote interface - must be shared between client and server.
 * All methods must throw RemoteException.
 * All parameters and return types must be either primitives or Serializable.
 *
 * Any object that is a remote object must implement this interface.
 * Only those methods specified in a "remote interface" are available remotely.
 */
public interface IRemoteMath extends Remote {

    public double add(double a, double b) throws RemoteException;

    public double subtract(double a, double b) throws RemoteException;

}
```

Remote Interface

```java
package client;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import remote.IRemoteMath;

/**
 * This class retrieves a reference to the remote object from the RMI registry. It
 * invokes the methods on the remote object as if it was a local object of the type of the
 * remote interface.
 *
 */
public class MathClient {

    public static void main(String[] args) {

        try {
            //Connect to the rmiregistry that is running on localhost
            Registry registry = LocateRegistry.getRegistry("localhost");

            //Retrieve the stub/proxy for the remote math object from the registry
            IRemoteMath remoteMath = (IRemoteMath) registry.lookup("Compute");

            //Call methods on the remote object as if it was a local object
            double addResult = remoteMath.add(5.0, 3.0);
            System.out.println("5.0 + 3.0 = " + addResult);
            double subResult = remoteMath.subtract(5.0, 3.0);
            System.out.println("5.0 - 3.0 = " + subResult);

        }catch(Exception e) {
            e.printStackTrace();
        }

    }

}
```

MathClient

```java
package server;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import remote.IRemoteMath;

/**
 * Creates an instance of the RemoteMath class and
 * publishes it in the rmiregistry
 *
 */
public class RMIServer {

    public static void main(String[] args)  {

        try {

            //Export the remote math object to the Java RMI runtime so that it
            //can receive incoming remote calls.
            //Because RemoteMath extends UnicastRemoteObject, this
            //is done automatically when the object is initialized.
            IRemoteMath remoteMath = new RemoteMath();

            //Publish the remote object's stub in the registry under the name "Compute"
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Compute", remoteMath);

            System.out.println("Math server ready");

            //The server will continue running as long as there are remote objects exported into
            //the RMI runtime, to remove remote objects from the
            //RMI runtime so that they can no longer accept RMI calls you can use:
           // UnicastRemoteObject.unexportObject(remoteMath, false);

        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

RMIServer

```java
package server;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import remote.IRemoteMath;

/**
 * Server side implementation of the remote interface.
 * Must extend UnicastRemoteObject, to allow the JVM to create a
 * remote proxy/stub.
 *
 */
public class RemoteMath extends UnicastRemoteObject implements IRemoteMath {

    private int numberOfComputations;

    protected RemoteMath() throws RemoteException {
        numberOfComputations = 0;
    }

    @Override
    public double add(double a, double b) throws RemoteException {
        numberOfComputations++;
        System.out.println("Number of computations performed so far = "
                + numberOfComputations);
        return (a+b);
    }

    @Override
    public double subtract(double a, double b) throws RemoteException {
        numberOfComputations++;
        System.out.println("Number of computations performed so far = "
                + numberOfComputations);
        return (a-b);
    }

}
```

RemoteMath

7_Security

1. List and briefly explain some worst case assumptions when designing a secure system.

- Networks are insecure.
    - Messages can be looked at, copied, modified and retransmitted.
    - Attackers can obtain information that they should not and can pretend to be a legitimate party.

- The source code is known to the attacker.
    - Knowing the source code can help the attacker discover vulnerabilities.
- Interfaces are exposed
    - Communication interfaces are necessarily open to allow clients to access them.
    - Attackers can send messages to any interface.
- The attacker has unlimited computing resources.
    - Assume that attackers will have access to the largest and most powerful computers projected in the lifetime of a system.

2. Define encryption and describe the two main types of keys used by encryption algorithms.

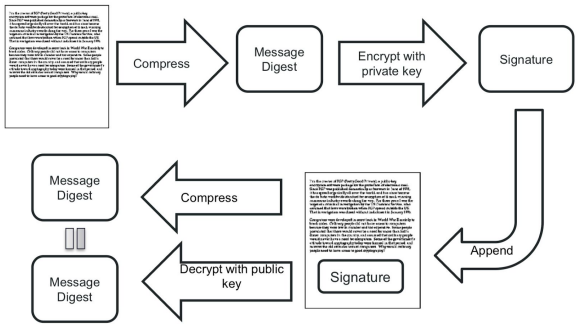
- Encryption
    - process of encoding a message in such a way as to hide its contents.
- **Shared secret keys (symmetric)**
    - Sender and recipient share knowledge of the key and it must not be revealed to anyone else.
- **Public/private key pairs (asymmetric)**
    - The sender uses a public key to encrypt the message.
    - The recipient uses a corresponding private key to decrypt the message.
    - Only the recipient can decrypt the message, because they have the private key.
    - Typically require 100 to 1000 times as much processing power as secret-key algorithms.

3. Discuss the three major roles that encryption plays in the implementation of secure systems.

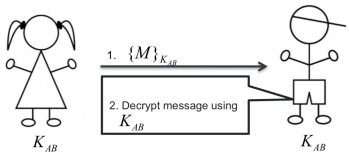
- **Secrecy保密 and integrity**
    - Messages encrypted with a particular key can only be decrypted by a recipient who knows the corresponding decryption key => Secrecy.
    - Integrity can be maintained if some redundant information such as a checksum is included and checked in the encrypted message.
- **Authentication**
    - If keys are held in private, a successful decryption authenticates the decrypted message as coming from a particular sender.
- **Digital signatures**
    - Verify to a third party that a message or a document is an unaltered copy of one produced by the signer.
    - It is a "stamp" Bob places on the data which is unique to Bob, and is very difficult to forge伪造.
    - It also assures保证 that any changes made to the data that has been signed cannot go undetected. 已经被签名的数据被改变也能被查到

4. Explain how digital signatures work.

(1) To sign a document, Bob first ‘compresses’ the message into just a few lines. This is called a digest文摘.
(2) Bob then encrypts the message digest with his private key. The result is the digital signature.
(3) Bob appends the digital signature to the document. All of the data that was ‘compressed’ into the digest has been signed.
(4) Bob sends the document to Alice.
(5) Alice decrypts the signature (using Bob's public key) changing it back into a message digest.
(6) Alice ‘compresses’ the document data into a message digest. If the message digest is the same as the message digest created when the signature was decrypted, then Alice knows that the signed data has not been changed and that Bob signed the document (because only Bob has his private key)

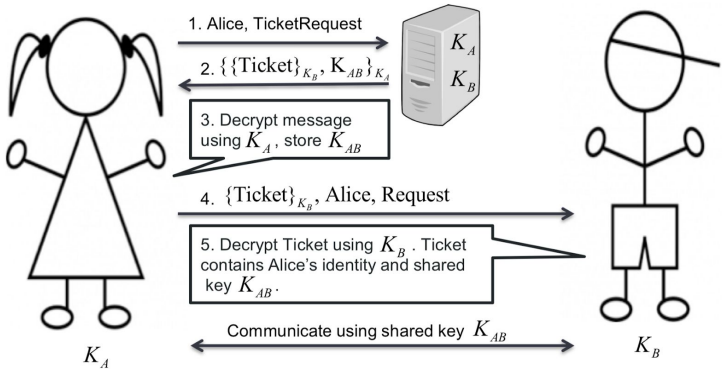


5. How does Alice send a secret message to Bob if they both share a secret key?



(1) Alice uses K_AB and an agreed encryption function E(K_AB, M) to encrypt and send the message to Bob.
(2) Bob decrypts the encrypted message using the corresponding decryption function D(K_AB, M).

6. How can Alice authenticate and communicate secretly with Bob assuming there is an authentication server that knows Alice’s and Bob’s secret keys?



Bob is a file server; Sara is an authentication service. Sara shares secret key K_A with Alice and secret key K_B with Bob.

1. Alice sends an (unencrypted) message to Sara stating her identity and requesting a *ticket* for access to Bob. ➜
2. Sara sends a response to Alice. $\{\{Ticket\}_{K_B}, K_{AB}\}_{K_A}$. It is encrypted in K_A and consists of a ticket (to be sent to Bob with each request for file access) encrypted in K_B and a new secret key K_{AB}.
3. Alice uses K_A to decrypt the response.
4. Alice sends Bob a request R to access a file: $\{Ticket\}_{K_B}$, Alice, R.
5. The ticket is actually $\{K_{AB}, Alice\}_{K_B}$. Bob uses K_B to decrypt it, checks that Alice's name matches and then uses K_{AB} to encrypt responses to Alice.

- This is a simplified version of the Needham and Schroeder (and Kerberos) protocol
- Timing and replay issues – addressed in N-S and Kerberos.
- Not suitable for e-commerce because authentication service doesn't scale…

Sara即认证服务器维护一张表，为系统所知的每一个主题保存一个名字和一个密钥，这个密钥只用于认证连接到认证服务器的客户进程，保证客户进程与认证服务器之间安全地进行信息传递，该密钥不泄露给第三方，生成后在网络上最多传送一次(理想情况下应以网络以外的途径传送)
在上图中K_A是Alice和Sara约定好的密钥，K_B同理
在第2步中，S返回给Alice用K_A加密的消息，消息含有新生成的密钥K_{AB}和一个用K_B加密的票证
第3步，Alice用K_A密钥解密
第4步，Alice把用K_B加密好的票证和K_{AB}以及需求R发给Bob
第5步，Bob用K_B解密票证，并使用新的密钥K_{AB}加密回复信息并返回给Alice
之后A和B共有了K_{AB}，除AB之外只有可信赖的S知道K_{AB}

以下非Workshop

7. Security Threats - Three broad Classes

– Leakage泄露: Acquisition获得 of information by unauthorised recipients
– Tampering篡改: Unauthorised alteration of information
– Vandalism恶意破坏: Interference with the proper operation of systems

8. Method of Attacks

(1) Eavesdropping窃听 - A form of leakage
(2) Masquerading伪装 – A form of impersonating冒充
(3) Message tampering篡改
(4) Replaying –storing secure messages and sending them at a later date
(5) Denial of service - Vandalism–flooding a channel or other resource, denying access to others

9. Limitation of Needham and Schroeder Protocols

● It depends upon prior knowledge by the authentication server Sara of Alice's and Bob's keys. This is feasible in a single organisation where Sara run a physically secure computer and is managed by a trusted principal. 依靠Sara必须在可信赖的环境下

–Not suitable in E-commerce or other wide area applications.

● Usefulness of challenges: They introduced the concept of a cryptographic challenge. That means in step 2 of our scenario, where Sara issues a ticket to Alice encrypted in Alice's secret key, K_A.

9. Authenticated communication with public keys

Bob has a public/private key pair <K_{Bpub}, K_{Bpriv}> & establishes K_{AB} as follows:

1. Alice obtains a certificate that was signed by a trusted authority stating Bob's public key K_{Bpub}
2. Alice creates a new shared key K_{AB} , encrypts it using K_{Bpub} using a public-key algorithm and sends the result to Bob.
3. Bob uses the corresponding private key K_{Bpriv} to decrypt it.

(If they want to be sure that the message hasn't been tampered with, Alice can add an agreed value to it and Bob can check it.)

10. Certificates - a statement signed by an appropriate authority.

Certificates require:

• An agreed standard format

• Agreement on the construction of chains of trust.

• Expiry dates, so that certificates can be revoked.

Alice's bank account certificate

1. *Certificate type*	Account number
2. *Name*:	Alice
3. *Account*:	6262626
4. *Certifying authority*:	Bob's Bank
5. *Signature*:	$\{Digest(field\ 2 + field\ 3)\}_{K_{Bpriv}}$

Public-key certificate for Bob's Bank

1. *Certificate type*	Public key
2. *Name*:	Bob's Bank
3. *Public key*:	K_{Bpub}
4. *Certifying authority*:	Fred – The Bankers Federation
5. *Signature*:	$\{Digest(field\ 2 + field\ 3)\}_{K_{Fpriv}}$

11. Access control Two main approaches to implementation:

–Access control list (ACL) associated with each object

E.g. Unix file access permissions

For more complex object types and user communities, ACLs can complex

```
drwxr-xr-x  gfc22  staff       264 Oct 30 16:57 Acrobat User Data
-rw-r--r--  gfc22  unknown       0 Nov  1 09:34 Eudora Folder
-rw-r--r--  gfc22  staff    163945 Oct 24 00:16 Preview of xx.pdf
drwxr-xr-x  gfc22  staff       264 Oct 31 13:09 iTunes
-rw-r--r--  gfc22  staff       325 Oct 22 22:59 list of broken apps.rtf
```

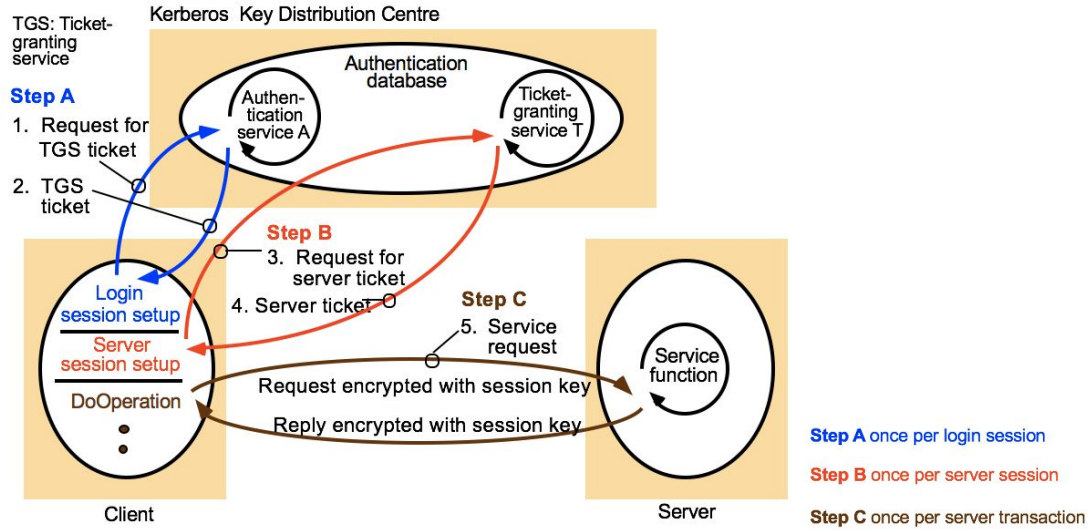
–Capabilities associated with principals

Like a key – allowing the holder access to certain operations on a specified resource.

Format: <resource id, permitted operations, authentication code>

12. Kerberos

一个Kerberos服务器被称为一个密钥分发中心(KDC)，每个KDC提供认证服务(AS)和票证授予服务(TGS)，AS用网络安全的口令认证用户，然后给代表用户的进程授予一张tickethe用来通信的会话密钥(session key)



Step A 认证你是你自己(client用ID和密码登录):
(1) Client请求认证(AS)服务器A提供TGS ticket (明文请求)
(2) A返回给C一条消息，包括用K_A加密的票证和与T(票证受益于服务器TGS)的会话密钥K_{CT}，用K_C加密确保C的身份
Step B 为服务器S获取票证
(3) C请求与服务器S通信的票证
(4) T检查C的ticket(C,T), 若有效则返回新的随机会话密钥K_{CS},并用K_S加密票证一起返回
Step C 用之前获得的K_{CS}与Server对话
(5) C向S发票证，附上用K_S加密的ticket，之后数据可用K_{CS}加密
(6) S可用K_S解密获取K_{CS}

Kerberized NFS(Network File System)

Kerberos protocol is too costly to apply on each NFS operation
Kerberos is used in the mount安装 service:
- to authenticate认证 the user's identity
- User's UserID and GroupID are stored at the server with the client's IP address

For each file request:
- UserID and GroupID are sent encrypted in the shared session key
- The UserID and GroupID must match those stored at the server
- IP addresses must also match

This approach has some problems
- can't accommodate容纳 multiple users sharing the same client computer
- all remote filestores must be mounted each time a user logs in

13. TLS/SSL The Secure Socket Layer

Extended and adopted as an Internet standard with the name Transport Level Security (TLS)
Supported by most browsers and is widely used in Internet commerce.

Key Feature

Negotiable encryption and authentication algorithms.协商加密和认证算法 In an open network we should NOT assume that all parties use the same client software or all client/server software includes a particular encryption algorithms.

Design requirements

– Secure communication without prior negotiation or help from 3rd parties 初始通信不加密
– Free choice of crypto algorithms by client and server
– Communication in each direction can be authenticated, encrypted or both

Bootstrapped secure communication 自举安全通信

● To meet the need for secure communication without previous negotiation/help from 3rd parties, the secure channel is established using a hybrid schemes. 为了不经过事先协商或第三方帮助进行不加密的初始通信，用混合方案建立安全信道

● The secure channel is fully configurable. 完全可配置

● The details of TLS protocols are standardized and several software libraries and toolkits are available to support it [www.openssl.org]

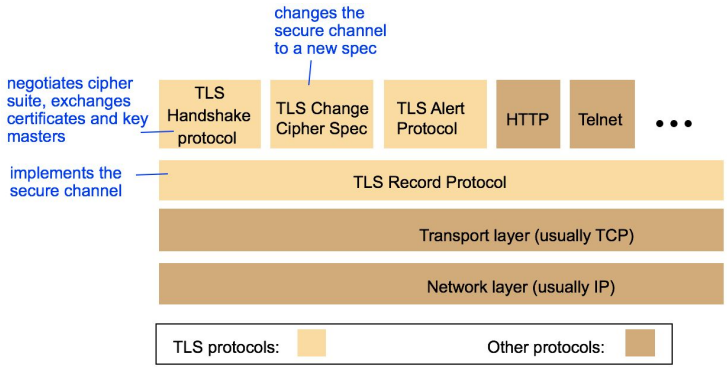
● TLS consists of two layers由两层组成:

– TLS Record Protocol Layer记录协议层: implements a secure channel, encrypting and authenticating messages transmitted through any connection oriented protocol. It is realized at session layer. 实现安全信道，用来加密和认证通过任何面向连接的协议传输的消息

– Handshake Layer握手层: Containing Handshake protocol and two other related protocols that establish and maintain a TLS session (i.e., secure channel) between client and server. 包括TLS握手协议和两个相关协议，他在客户和服务器间建立并维护一个TLS会话

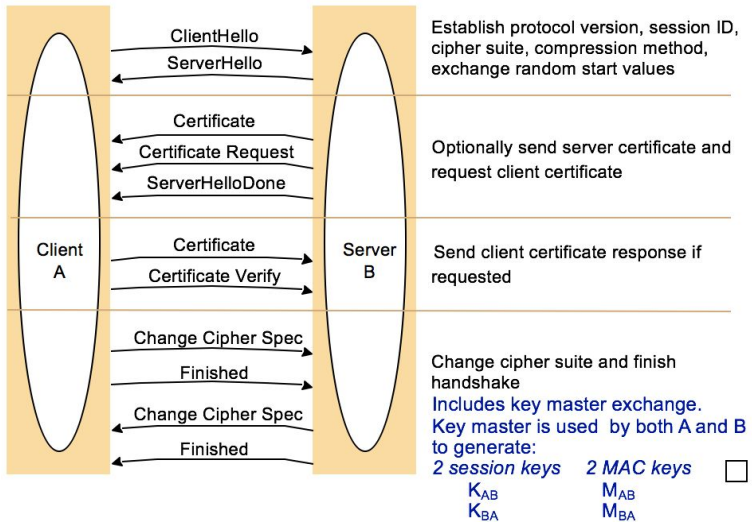
– Both are implemented by software libraries at application level in the client and the server. 都是用客户和服务器应用层的软件库实现的

TLS protocol stack



TLS/SSL handshake protocol

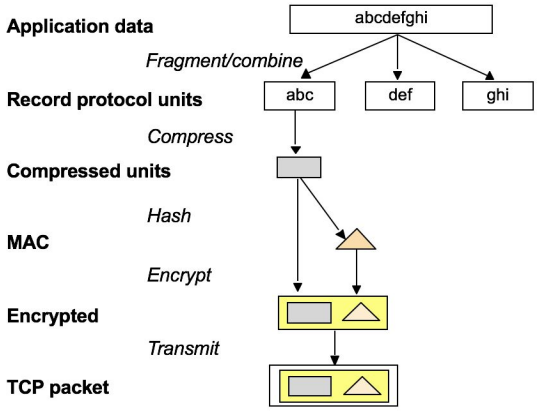
(Handshake is performed over an existing connection)



TLS handshake configuration options

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA (International Data Encryption Algorithm)
Message digest function	for creating message authentication codes (MACs)	SHA (Secure Hash Algorithm)

TLS record protocol operation: a pipeline for data transformation



Encryption is the process of encoding a message in such a way as to hide its contents.
A **cryptography key** is a parameter used in an encryption algorithm in such a way that the encryption cannot be reversed without a knowledge of the key.

Tutorial 9

14. What is a digital certificate数字证书 and why do we need it?

- A digital certificate is a digital form of identification, like a passport.
- A digital certificate provides information about the identity of an entity.
- A digital certificate is issued by a Certification Authority (CA).
    - Examples of trusted CA across the world are Verisign, Entrust, etc.

    - The CA guarantees the validity of the information in the certificate.
- The issue of distributing Public Key is massive, because the Public Key should be distributed in a scalable and truthful way

15. Public Key Infrastructure (PKI) 公开密钥基础建设

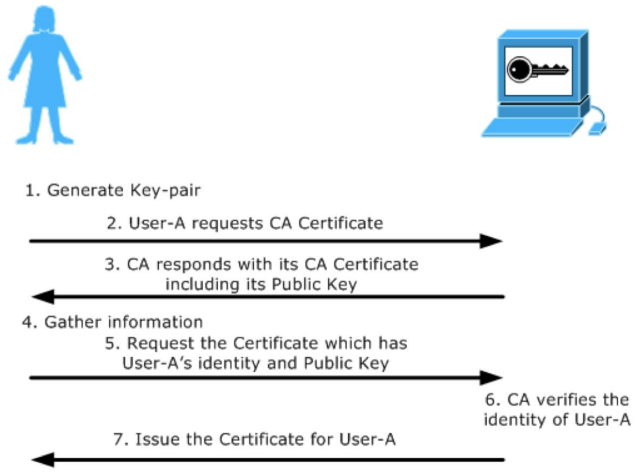
- Public Key Infrastructure (PKI) consists of protocols, standards and services, that allows users to authenticate each other using digital certificates that are issued by CA. For a digital certificate to be useful, it has to be structured in a standard way so that information within the certificate can be retrieved and understood regardless of who issued the certificate. The X.509, PKI X.509 and Public Key Cryptography Standards (PKCS) are the building blocks a PKI system that defines the standard formats for certificates and their use.

16. Certificates

Certificate type: Public key
Name: Bob
Public key: kBpub
Certifying authority: Sara
Signature: {Digest(field 2+field 3)}_{kSpriv}

- In your own words, what is this certificate saying?
    - Sara certifies that Bob's public key is kBpub
- Why can't Sara deny that she has attested证明 to this fact?
    - Because if someone can decrypt解码 the signature using kSpub, only someone who had kSpriv could have encrypted it.
- What must be known to anyone who wants to make sure the certificate is authentic?
    - kSpub

17. What is the process to obtain a digital certificate?



1. Generate Key-pair: User-A generates a Public and Private key-pair or is assigned a key-pair by some authority in their organization.

2. Request CA Certificate: User-A first requests the certificate of the CA Server.

3. CA Certificate Issued: The CA responds with its Certificate. This includes its Public Key and its Digital Signature signed using its Private Key.

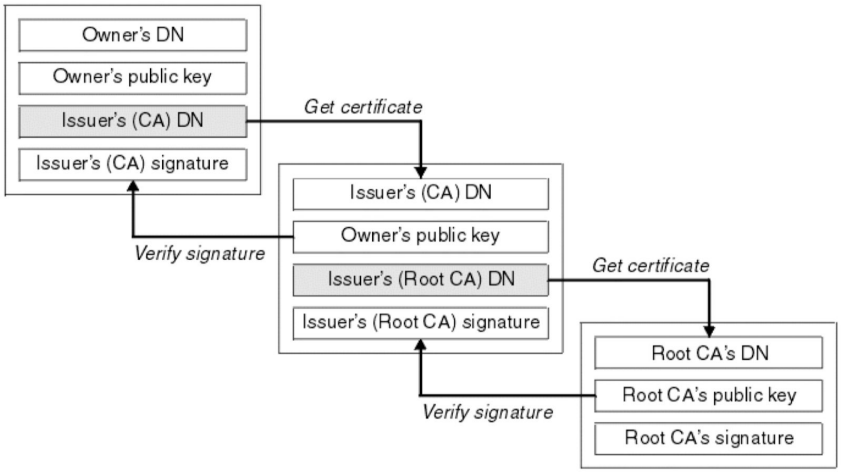
4. Gather Information: User-A gathers all information required by the CA Server to obtain its certificate. This information could include User-A email address, fingerprints, etc. that the CA needs to be certain that User-A claims to be who she is.

5. Send Certificate Request: User-A sends a certificate request to the CA consisting of her Public Key and additional information. The certificate request is signed by CA's Public Key.

6. CA verifies User-A: The CA gets the certificate request, verifies User-A's identity and generates a certificate for User-A, binding her identity and her Public Key. The signature of CA verifies the authenticity of the Certificate.

7. CA issues the Certificate: The CA issues the certificate to User-A.

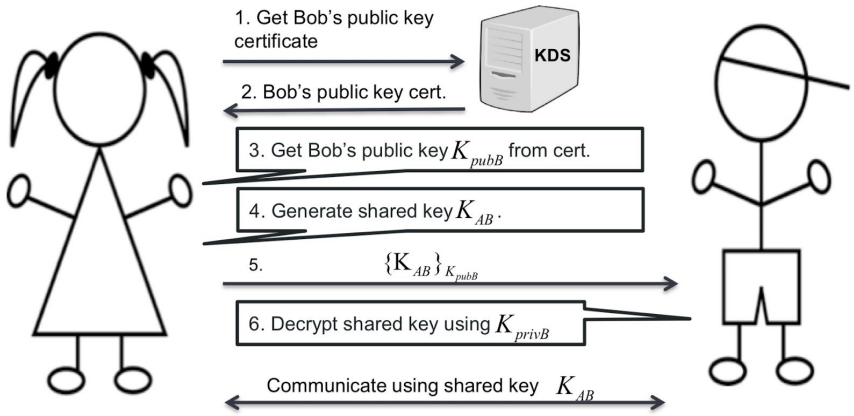
18. What is a certificate chain?



Example usage of certificate chain

● During a User's certificate validation by a browser or a program, browser needs to validate the signature by finding the public key of the next issuing CA or intermediate CA. The process will continue until the root certificate is reached. Root CA is self signed and must be trusted by the browser at the end. Browsers keep all well known CAs root certificates in their trust store.

19. Assuming that Bob has a public/private key pair, how can Alice and Bob establish a shared key to communicate secretly using a Key Distribution Service?



8_Distributed File Systems

1. What are some key design issues for distributed file systems?

• Effective use of client caching to achieve performance
• Maintain consistency between multiple copies of cached files
• Recovery after client or server failure
• High throughput for reading and writing files of all sizes
• Scalability

2. Name and explain three transparencies that should be addressed by distributed file systems.

Access transparency
    • Client programs don't know if the file is local of remote
Location transparency
    • Client programs don't know where the file is stored
    • Files can be relocated without changing their pathname路径
Mobility流动性 transparency
    • Neither client programs nor system administration管理 tables in client nodes need to be changed when files are moved
Performance transparency
    • Maintain acceptable performance while the load on the service varies within a specified range
Scaling transparency
    • Service can be expanded without loss of performance

3. What are the advantages and disadvantages of using absolute names as a naming strategy?

• Absolute name
    • provides a complete address to a file including both the server and path names: <machine name: path name>
• Advantages

    - Trivial to find a file once the name is given
    - No additional state must be kept since each name is self contained (No global state)
    - Greater scalability
    - Easy to add and delete new names
- Disadvantages
    - No location transparency
    - File is location dependent and cannot be moved
    - Less resilient弹性 to failure

4. What are the advantages and disadvantages of a naming strategy based on mount points?

- Mount Points (used by Sun's Network File System - NFS)
    - The client machine creates a set of "local names" which are used to refer to remote locations: mount points
    - At boot time, the local name is bound to the remote name
    - The operating system must maintain a table to maintain the mapping of what server and path are mapped to each mount point
- Advantages:
    - Names do not contain information about the file location
    - Remote location can change between reboots
- Disadvantages:
    - Hard to maintain
        - What happens when machines fail?
        - What happens when files are migrated?
    - Can lead to confusion since two different local names may map to the same file on a remote system

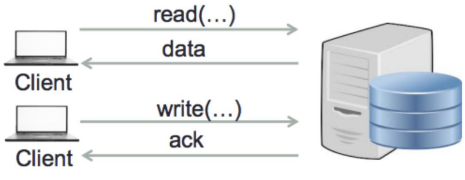
5. What are the advantages and disadvantages of using a global name space naming strategy?

- Global Name Space
    - All nodes have an identical name space: the path and name of a file on one machine will be the same on every other machine, regardless of where the file is actually stored
    - Dedicated file servers
    - Client contacts one of the servers and receives the layout of the distributed file system
    - When a user accesses a file, the server sends a copy of it to the client machine where it is cached
- Advantages
    - Location transparency
    - Naming is consistent across all clients
    - Storage servers are able to seamlessly move files around because clients always contact the server to learn where files are located
- Disadvantages
    - Files are cached by clients = challenges in keeping files consistent

- Can lead to performance problems, particularly when the scale of the system grows

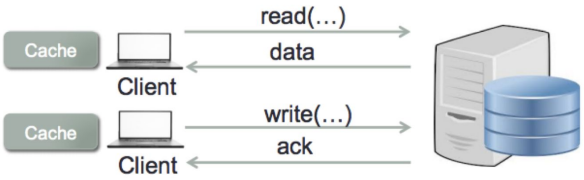
6. What are the advantages and disadvantages of the following simple distributed file system?

(1)



- RPC to access file system calls
- No client/local caching
- Advantages
    - Consistent view of file system
- Disadvantages
    - Performance (network access, server becomes potential bottleneck)

(2)



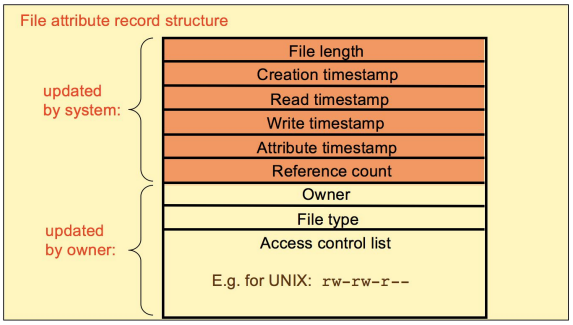
- Cache files on clients and perform local file system operations
- Advantages
    - Local operations = better performance
- Disadvantages
    - What happens when the client fails?
    - Where should the client store the cached files?
    - Difficult to keep local copy consistent with remote copy

7. Discuss the advantages and disadvantages of storing client cache files on disk and on memory?

- Disk
- Advantages:
    - Safer if client fails (more reliability)
- Disadvantages:
    - Slower than just keeping it in local memory
    - Requires client to have a disk
- Memory
- Advantages:
    - Quick access time
    - Works with diskless workstations
- Disadvantages:
    - Does not tolerate node failure well
    - Limited cache size

8. What is a file system?

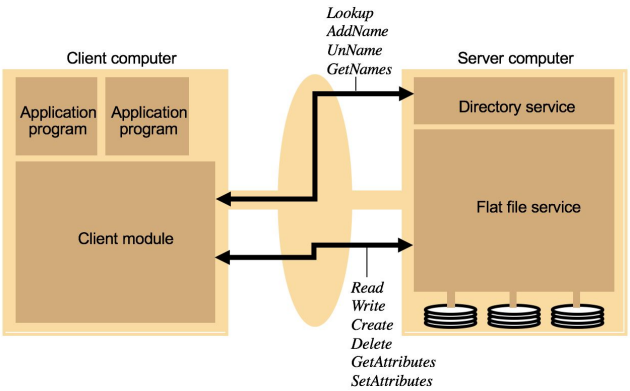
• Persistent stored data sets
• Hierarchic name space visible to all processes
• API with the following characteristics:
  – access and update operations on persistently stored data sets
  – Sequential access model (with additional random facilities)
• Sharing of data between users, with access control
• Concurrent access:
  – certainly for read-only access
  – what about updates?
• Other features:
  –mountable file stores



9. File Service Architecture

• An architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components:
  – A flat file service
  – A directory service
  – A client module
• The relevant modules and their relationship is (shown next).
• The Client module implements exported interfaces by flat file and directory services on server side.

10. Model file service architecture



The Client module implements nearly all of the state-dependent functionality (open files, position of read-write pointer, etc.)- supporting fault tolerance - and it maintains a cache of recently-used file data - efficiency, limited replication and scalability.
It also translates the file operations performed by applications into the API of the file server - transparency and heterogeneity. It supplies authentication information and performs encryption when necessary, but access control checks must be made by the server.

11. Responsibilities of various modules

- Flat file service:
  - – Concerned with the implementation of operations on the contents of file. Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat file service operations. UFIDs are long sequences of bits chosen so that each file has a unique among all of the files in a distributed system.
- Directory Service:
  - – Provides mapping between text names for the files and their UFIDs. Clients may obtain the UFID of a file by quoting its text name to directory service. Directory service supports functions needed to generate directories and to add new files to directories.
- Client Module:
  - – It runs on each computer and provides integrated service (flat file and directory) as a single API to application programs. For example, in UNIX hosts, a client module emulates the full set of Unix file operations.
  - – It holds information about the network locations of flat-file and directory server processes; and achieve better performance through implementation of a cache of recently used file blocks at the client.

12. Server operations/interfaces for the model file service

Flat file service

Read(FileId, i, n) -> Data (i: position of first byte)

Write(FileId, i, Data) (i: position of first byte)

Create() -> FileId

Delete(FileId)

GetAttributes(FileId) -> Attr

SetAttributes(FileId, Attr)

Directory service

Lookup(Dir, Name) -> FileId

AddName(Dir, Name, ~~File~~) FileId

UnName(Dir, Name)

GetNames(Dir, Pattern) -> NameSeq

Pathname lookup

Pathnames such as '/usr/bin/tar' are resolved by iterative calls to *lookup()*, one call for each component of the path, starting with the ID of the root directory '/' which is known in every client.

13. File Group

A collection of files that can be located on any server or moved between servers while maintaining the same names.

- Similar to a UNIX *filesystem*
- Helps with distributing the load of file serving between several servers.
- File groups have identifiers which are unique throughout the system (and hence for an open system, they must be globally unique).
  - *Used to refer to file groups and files*

To construct a globally unique ID we use some unique attribute of the machine on which it is created, e.g. IP number, even though the file group may move subsequently.

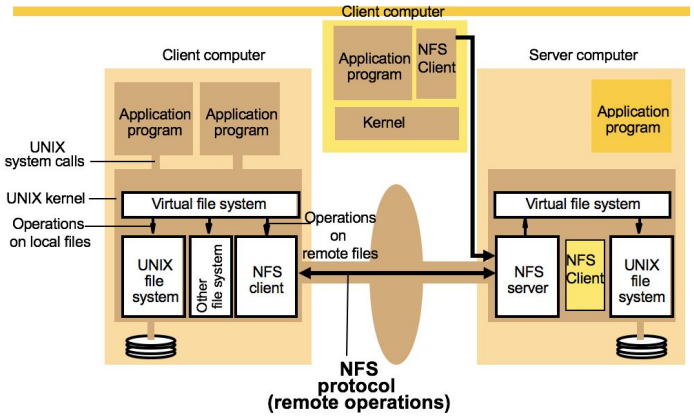
File Group ID:

32 bits	*16 bits*
IP address	date

14. Case Study: Sun NFS

- An industry standard for file sharing on local networks since the 1980s
- An open standard with clear and simple interfaces
- Closely follows the abstract file service model defined above
- Supports many of the design requirements already mentioned:
  - transparency
  - heterogeneity
  - efficiency
  - fault tolerance
- Limited achievement of:
  - concurrency
  - replication
  - consistency
  - security

NFS architecture



NFS architecture: does the implementation have to be in the system kernel?

No:

- there are examples of NFS clients and servers that run at application-level as libraries or processes (e.g. early Windows and MacOS implementations, current PocketPC, etc.)

But, for a Unix implementation there are advantages:

- Binary code compatible - no need to recompile applications
  - *Standard system calls that access remote files can be routed through the NFS client module by the kernel*
- Shared cache of recently-used blocks at client
- Kernel-level server can access i-nodes and file blocks directly
  - *but a privileged (root) application program could do almost the same.*
- Security of the encryption key used for authentication.

NFS access control and authentication

- Stateless server, so the user's identity and access rights must be checked by the server on each request.
  - In the local file system they are checked only on *open()*
- Every client request is accompanied by the userID and groupID
  - which are inserted by the RPC system
- Server is exposed to imposter attacks unless the userID and groupID are protected by encryption
- Kerberos has been integrated with NFS to provide a stronger and more comprehensive security solution

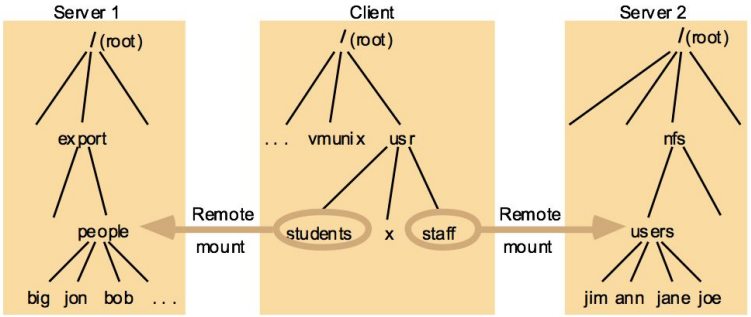
Architecture Components (UNIX / Linux)

- Server:
  - nfsd: NFS server daemon that services requests from clients.
  - mountd: NFS mount daemon that carries out the mount request passed on by nfsd.
  - rpcbind: RPC port mapper used to locate the nfsd daemon.
  - /etc/exports: configuration file that defines which portion of the file systems are exported through NFS and how.
- Client:
  - mount: standard file system mount command.
  - /etc/fstab: file system table file.
  - nfsiod: (optional) local asynchronous NFS I/O server.

Mount service

- Mount operation:
  mount(remotehost, remotedirectory, localdirectory)
- Server maintains a table of clients who have mounted filesystems at that server
- Each client maintains a table of mounted file systems holding:
  < IP address, port number, file handle>
- *Hard* versus *soft* mounts

Local and remote file systems accessible on an NFS client



Note: The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1; the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

Automounter

NFS client catches attempts to access 'empty' mount points and routes them to the Automounter
- Automounter has a table of mount points and multiple candidate serves for each
- it sends a probe message to each candidate server and then uses the mount service to mount the filesystem at the first server to respond

- Keeps the mount table small
- Provides a simple form of replication for read-only filesystems
  - E.g. if there are several servers with identical copies of /usr/lib then each server will have a chance of being mounted at some clients.

Kerberized NFS

- Kerberos protocol is too costly to apply on each file access request
- Kerberos is used in the mount service:
  - to authenticate the user's identity
  - User's UserID and GroupID are stored at the server with the client's IP address
- For each file request:
  - The UserID and GroupID sent must match those stored at the server
  - IP addresses must also match
- This approach has some problems
  - can't accommodate multiple users sharing the same client computer
  - all remote filestores must be mounted each time a user logs in

New design approaches

Distribute file data across several servers
- Exploits high-speed networks (InfiniBand, Gigabit Ethernet)
- Layered approach, lowest level is like a 'distributed virtual disk'
- Achieves scalability even for a single heavily-used file

'Serverless' architecture
- Exploits processing and disk resources in all available network nodes
- Service is distributed at the level of individual files

Examples:

xFS : Experimental implementation demonstrated a substantial performance gain over NFS and AFS

Peer-to-peer systems: Napster, OceanStore (UCB), Farsite (MSR), Publius (AT&T research) - see web for documentation on these very recent systems

Cloud-based File Systems: **DropBox**

9_Name Services

1. Define and discuss the concepts of Uniform Resource Identifiers

- Uniform Resource Identifiers (URIs) are concerned with identifying resources on the Web, and other Internet resources such as electronic mailboxes.
- URIs are intended to allow a generic way of specifying the identifier so as to make it easy for common software to process the identifier. This allows new types of identifiers to be readily introduced and for existing identifiers to be used by a wide variety of different software and services.
- URLs provide ways to locate the resource being named. They clearly suffer if the resource has since changed its name (e.g. broken links in the Web).

2. What are Name Services and why do we need them?

- Definition

  In a Distributed System, a Naming Service is a specific service whose aim is to provide a consistent and uniform naming of resources, thus allowing other programs or services to localize them and obtain the required metadata for interacting with them.
- Key benefits

  – Resource localization (An URL facilitates the localization of a resource exposed on the Web. e.g., abc.net.au means it is likely to be an Australian entity)

– Uniform naming (A consistent and uniform naming helps processes in a distributed system to interoperate and manage resources. e.g., commercials use .com; non-profit organizations use .org)
– Device independent address (e.g., you can move domain name/web site from one server to another server seamlessly).

3. What is the process of name resolution?

- Name resolution is in general an iterative process. A name either resolves to a set of primitive attributes or it resolves to another name.
- The use of aliases make it possible for resolution cycles to occur and the potential for the resolution process to never terminate. Two solutions to overcome this include:
    - Abandon the resolution process after some number of iterations.
    - Require administrators to ensure that no cycles occur

4. What is navigation and what are the approaches to navigation?

- When the name service is distributed then a single server may not be able to resolve the name. The resolve request may need to propagate from one server to another, referred to as *navigation*.
- Classification of different navigation approaches.
    - *iterative navigation* -- The client makes the request at different servers one at a time. The order of servers visited is usually in terms of domain hierarchy. Always starting at the root server would put excessive load on the root.
    - *multicast navigation* -- The client multicasts the request to the group (or a subset) of name servers. Only the server that holds the named request returns a result.
    - *non-recursive server-controlled navigation* -- The client sends the request to a server and the server continues on behalf of the client, as above.
    - *recursive server-controlled navigation* -- The client sends the request to a server and the server sends the request to another server (if needed) recursively.

5. What is Domain Name System and in what aspects does it improve on a file-based implementation?

- The Domain Name System (DNS) is a name service design whose main naming database is used across the Internet.
- Before DNS, all host names and addresses were held in a single central master file and downloaded by FTP to all computers that required them.
- The problems with the original name service included:
    - It did not scale to large numbers of computers.
    - Local organizations wished to administer their own naming systems.
    - A general name service was needed -- not one that serves only for looking up computer addresses.
- DNS is designed for use in multiple implementations, each of which may have its own name space, though in practice the Internet DNS name space is the one in widespread use.

以下非Workshop

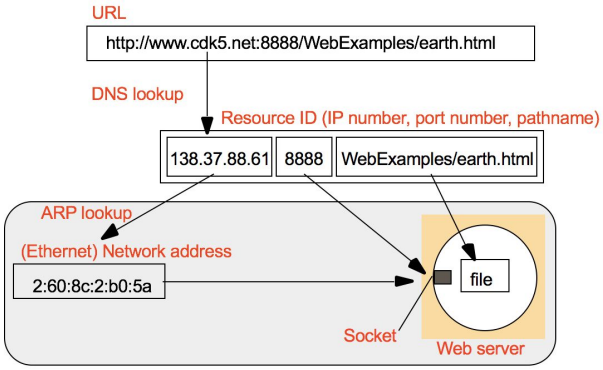
The role of names and name services

- Resources are accessed using *identifier* or *reference*
  - An identifier can be stored in variables and retrieved from tables quickly
  - Identifier includes or can be transformed to an address for an object
    - *E.g. NFS file handle, CORBA remote object reference*
  - A *name* is human-readable value (usually a string) that can be *resolved* to an identifier or address
    - *Internet domain name, file pathname, process number*
    - *E.g ./etc/passwd, http://www.cdk5.net/*
- For many purposes, names are preferable to identifiers
  - because the binding of the named resource to a physical location is deferred and can be changed
  - because they are more meaningful to users
- Resource names are *resolved* by name services
  - to give identifiers and other useful attributes

Requirements for name spaces

- Allow simple but meaningful names to be used
- Potentially infinite number of names
- Structured
  - to allow similar subnames without clashes
  - to group related names
- Allow re-structuring of name trees
  - for some types of change, old programs should continue to work
- Management of trust

Composed naming domains used to access a resource from a URL



Names and resources

- Currently, different name systems are used for each type of resource:

resource	*name*	*identifies*
file	pathname	file within a given file system
process	process id	process on a given computer
port	port number	IP port on a given computer

- Uniform Resource Identifiers (URI) offer a general solution for any type of resource. There two main classes:

  URL *Uniform Resource Locator (URL)*
  - typed by the protocol field (http, ftp, nfs, etc.)
  - part of the name is service-specific
  - resources cannot be moved between domains

  URN *Uniform Resource Name (URN)*
  - requires a universal resource name lookup service - a DNS-like system for all resources

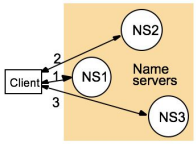
14

Navigation

Navigation is the act of chaining multiple Naming Services in order to resolve a single name to the corresponding resource.

- Namespaces allows for structure in names.
- URLs provide a default structure that decompose the location of a resource in
  - protocol used for retrieval
  - internet end point of the service exposing the resource
  - service specific path
- This decomposition facilitates the resolution of the name into the corresponding resource
- Moreover, structured namespaces allows for iterative navigation…

Iterative navigation



A client iteratively contacts name servers NS1–NS3 in order to resolve a name

Used in:

DNS: Client presents entire name to servers, starting at a local server, NS1. If NS1 has the requested name, it is resolved, else NS1 suggests contacting NS2 (a server for a domain that includes the requested name).

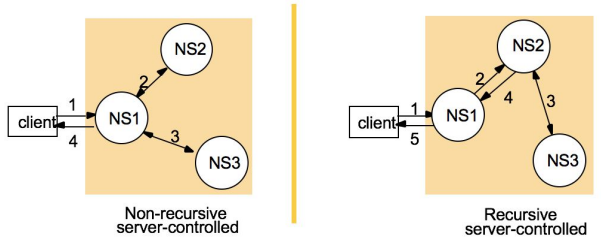
NFS: Client segments pathnames (into 'simple names') and presents them one at a time to a server together with the filehandle of the directory that contains the simple name.

Reason for NFS iterative name resolution
This is because the file service may encounter a symbolic link (i.e. an *alias*) when resolving a name. A symbolic link must be interpreted in the client's file system name space because it may point to a file in a directory stored at another server. The client computer must determine which server this is, because only the client knows its mount points.

Server controlled navigation

- In an alternative model, name server coordinates naming resolution and returns the results to the client. It can be:
  - Recursive:
    - *it is performed by the naming server*
    - *the server becomes like a client for the next server*
    - *this is necessary in case of client connectivity constraints*
  - Non recursive:
    - *it is performed by the client or the first server*
    - *the server bounces back the next hop to its client*

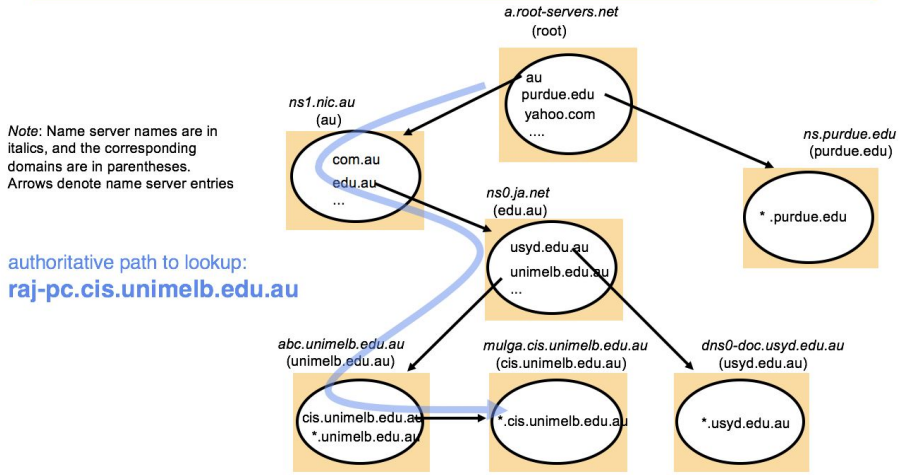
Non-recursive and recursive server-controlled navigation



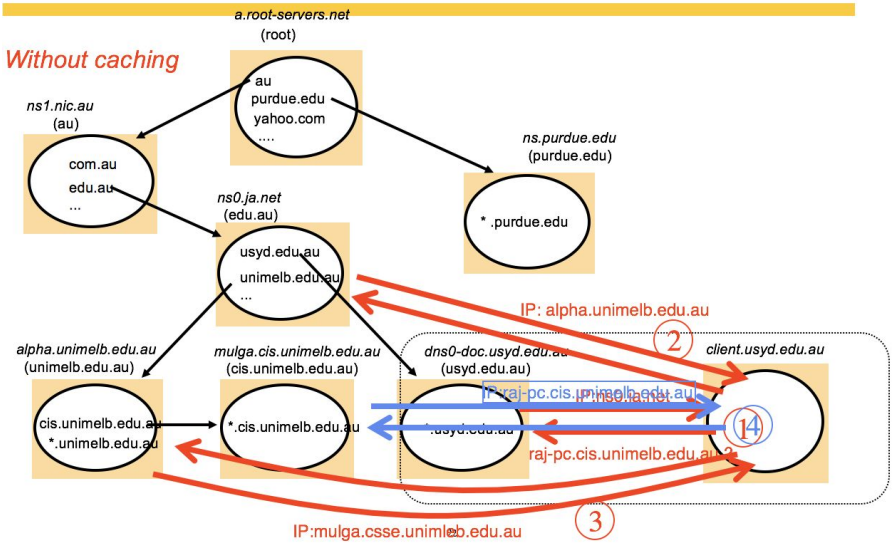
A name server NS1 communicates with other name servers on behalf of a client

DNS offers recursive navigation as an option, but iterative is the standard technique. Recursive navigation must be used in domains that limit client access to their DNS information for security reasons.

DNS name servers: Hierarchical organisation



DNS in typical operation



DNS server functions and configuration

- Main function is to resolve domain names for computers, i.e. to get their IP addresses
  - caches the results of previous searches until they pass their 'time to live'
- Other functions:
  - get *mail host* for a domain
  - reverse resolution - get domain name from IP address
  - Host information - type of hardware and OS
  - Well-known services - a list of well-known services offered by a host
  - Other attributes can be included (optional)

DNS resource records

Record type	*Meaning*	*Main contents*
A	A computer address (IPv4)	IPv4 number
AAA	A computer address (IPv6)	IPv6 number
NS	An authoritative name server	Domain name for server
CNAME	The canonical name for an alias	Domain name for alias
SOA	Marks the start of data for a zone	Parameters governing the zone
PTR	Domain name pointer (reverse lookups)	Domain name
HINFO	Host information	Machine architecture and operating system
MX	Mail exchange	List of <*preference*, *host*> pairs
TXT	Text string	Arbitrary text

DNS issues

- Name tables change infrequently, but when they do, caching can result in the delivery of stale data.
  - Clients are responsible for detecting this and recovering
- Its design makes changes to the structure of the name space difficult. For example:
  - merging previously separate domain trees under a new root
  - moving subtrees to a different part of the structure (e.g. if Scotland became a separate country, its domains should all be moved to a new country-level domain.)

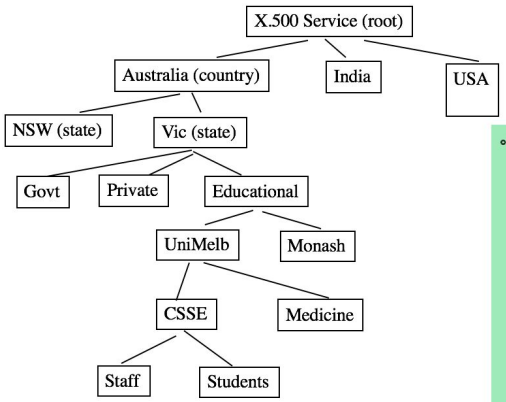
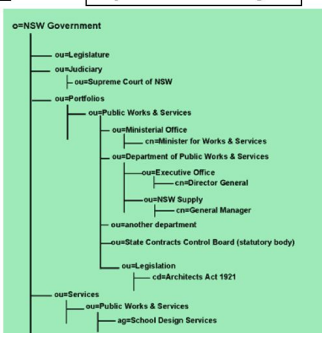
Directory services (registration and discovery)

- Sometime users wish to find a particular person or resource, but they don't know its name, only some of its attributes.
  - What is the name of the user with a telephone number **03-83441344**?
  - What is the name of professor teaching Cloud computing at UniMelb (e.g., **ask Google!**)
- Sometime users require a service, but they are not concerned with what system entity provides it.
  - Where can I print high resolution colour image?
- Directory services can help with above situation: they store collections of bindings and attributes and also looks up entries that match attribute-based specs.
- Directory service:- 'yellow pages' for the resources in a network
  - Retrieves the set of names that satisfy a given description
  - e.g. X.500, LDAP, MS Active Directory Services
    - (*DNS holds some descriptive data, but:*
      - the data is very incomplete
      - DNS isn't organised to search it)
- Discovery service:- a directory service that also:
  - is automatically updated as the network configuration changes
  - meets the needs of clients in spontaneous networks
  - discovers services required by a client (who may be mobile) within the current *scope*, for example, to find the most suitable printing service for image files after arriving at a hotel.
  - *Examples of discovery services*: Jini discovery service, the 'service location protocol', the 'simple service discovery protocol' (part of UPnP), the 'secure discovery service'.

X.500 Directory Service

- X.500 and LDAP (Lightweight Directory Access Protocol)
  - a hierarchically-structured standard directory service designed for world-wide use
  - X.500 is standardised by ITU (international telecommunication union) and ISO
  - accommodates resource descriptions in a standard form and their retrieval for any resource (online or offline)
  - never fully deployed, but the standard forms the basis for LDAP, the Lightweight Directory Access Protocol, which is widely used – IETF RFC 2251.
  - A secure access to directory through authentication is also supported.

Part of the X.500 Directory Information Tree (DIT)



Object class for NSW govt.