Unit 5: Naming

Contents

Names play an important role in all computer systems. They are used to share resources, to uniquely identify entities, to refer to locations, and more. An important issue with naming is that a name can be resolved to the entity it refers to. Name resolution thus allows a process to access the named entity. To resolve names, it is necessary to implement a naming system. The difference between naming in distributed systems and non-distributed systems lies in the way naming systems are implemented.

5.1 Name, Identifiers, and Addresses

A name in a distributed system is a string of bits or characters that is used to refer to an entity. An entity in a distributed system can be practically anything. Typical examples include resources such as hosts, printers, disks, and files. Other well-known examples of entities that are often explicitly named are processes, users, mailboxes, newsgroups, Web pages, graphical windows, messages, network connections, and so on.

Entities can be operated on. For example, a resource such as a printer offers an interface containing operations for printing a document, requesting the status of a print job, and the like. Furthermore, an entity such as a network connection may provide operations for sending and receiving data, setting quality-of-service parameters, requesting the status, and so forth.

To operate on an entity, it is necessary to access it, for which we need an **access point**. An access point is yet another, but special, kind of entity in a distributed system. The name of an access point is called an **address**. The address of an access point of an entity is also simply called an address of that entity.

An entity can offer more than one access point. As a comparison, a telephone can be viewed as an access point of a person, whereas the telephone number corresponds to an address. Indeed, many people nowadays have several telephone numbers, each number corresponding to a point where they can be reached. In a distributed system, a typical example of an access point is a host running a specific server, with its address formed by the combination of, for example, an IP address and port number (i.e., the server's transport-level address).

An entity may change its access points in the course of time. For example, when a mobile computer moves to another location, it is often assigned a different IP address than the one it had before. Likewise, when a person moves to another city or country, it is often necessary to change telephone numbers as well. In a similar fashion, changing jobs or Internet Service Providers, means changing your e-mail address.

A name for an entity that is independent from its addresses is often much easier and more flexible to use. Such a name is called **location independent**.

In addition to addresses, there are other types of names that deserve special treatment, such as names that are used to uniquely identify an entity. A true identifier is a name that has the following properties:

1. An identifier refers to at most one entity.
2. Each entity is referred to by at most one identifier.
3. An identifier always refers to the same entity (i.e., it is never reused).

Addresses and identifiers are two important types of names that are each used for very different purposes. In many computer systems, addresses and identifiers are represented in machine-readable form only, that is, in the form of bit strings. For example, an Ethernet address is essentially a random string of 48 bits. Likewise, memory addresses are typically represented as 32-bit or 64-bit strings.

Another important type of name is that which is tailored to be used by humans, also referred to as human-friendly names. In contrast to addresses and identifiers, a human-friendly name is generally represented as a character string. These names appear in many different forms. For example, files in Unix systems have character-string names that can generally be as long as 255 characters, and which are defined entirely by the user. Similarly, DNS names are represented as relatively simple case-insensitive character strings.

In principle, a **naming system** maintains a name-to-address binding which in its simplest form is just a table of (name, address) pairs. However, in distributed systems that span large networks and for which many resources need to be named, a centralized table is not going to work.

The types of naming systems used are:

1. **Flat Naming**: Resolvers identifiers to addresses.
2. **Structured Naming**: Resolves structured human-friendly names to addresses.
3. **Attribute-based Naming**: Resolves descriptive names to addresses.

Desirable features of a good naming system

A good naming system for a distributed system should have the feature described below:

1. **Location transparency**: It means that the name of an object should not reveal any hint as to the physical location of the object.
2. **Location independency**: For performance, reliability, availability and security reasons, distributed systems provide the facility of object migration that allows the movement and relocation of objects dynamically among the various nodes of a system.
3. **Scalability**: Distributed systems vary in size ranging from one with a few nodes to one with many nodes. Distributed systems are normally open systems, and their size changes dynamically.
4. **Uniform naming convention**: In many existing systems, different ways of naming objects called naming conventions, are used for naming different types of objects. For example, filenames typically differ from user names and process names.
5. **Multiple user-defined names for the same object**: For a shared object, it is desirable that different users of the object can use their own convenient names for accessing it. A naming system must provide the flexibility to assign multiple user-defined names to the same object.
6. **Group naming**: A naming system should allow many different objects to be identified by the same name. Such a facility is useful to support broadcast facility or to group objects for conferencing or other applications.
7. **Meaningful names**: A name can be simply any character string identifying some object. For users, meaningful names are preferred to lower level identifiers such as memory pointers, disk block numbers, or network addresses.

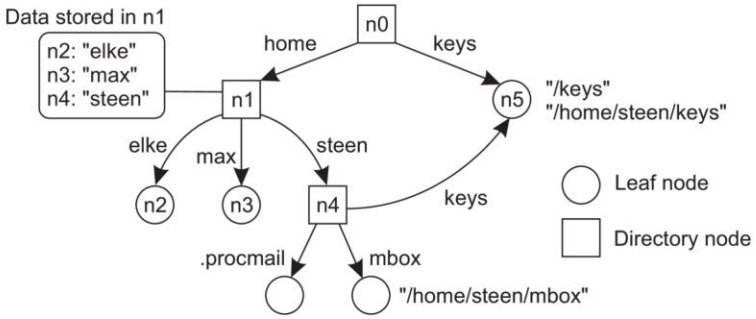
8. **Performance**: The most important performance measurement of a naming system is the amount of time needed to map an object's name to its attributes, such as its location. In a distributed environment, this performance is dominated by the number of messages exchanged during the name-mapping operation.
9. **Fault tolerance**: A naming system should be capable of tolerating, to some extent, faults that occur due to the failure of a node or a communication link in a distributed system network. The naming system should continue functioning, perhaps in a degraded form, in the event of these failures.
10. **Replication transparency**: In a distributed system, replicas of an object are generally created to improve performance and reliability.
11. **Locating the nearest replica**: When a naming system supports the use of multiple copies of the same object, it is important that the object-locating mechanism of the naming system should always supply the location of the nearest replica of the desired object.

5.2 Structured Naming

Flat names are good for machines, but are generally not very convenient for humans to use. As an alternative, naming systems generally support structured names that are composed from simple, human-readable names. Not only file naming, but also host naming on the Internet follows this approach.

Name spaces

Names are commonly organized into what is called a **name space**. Name spaces for structured names can be represented as a labeled, directed graph with two types of nodes.



A leaf node represents a named entity and has the property that it has no outgoing edges. A leaf node generally stores information on the entity it is representing–for example, its address–so that a client can access it. Alternatively, it can store the state of that entity, such as in the case of file systems in which a leaf node actually contains the complete file it is representing.

In contrast to a leaf node, a directory node has a number of outgoing edges, each labeled with a name, as shown in Figure. Each node in a naming graph is considered as yet another entity in a distributed system, and, in particular, has an associated identifier. A directory node stores a table in which an outgoing edge is represented as a pair (node identifier, edge label). Such a table is called a directory table.

The naming graph shown in Figure has one node, namely n0, which has only outgoing and no incoming edges. Such a node is called the **root** (node) of the naming graph. Although it is possible for a naming graph to have several root nodes, for simplicity, many naming systems have only one. Each path in a naming graph can be referred to by the sequence of labels corresponding to the edges in that path, such as N:[label1, label2, ..., labeln], where N refers to the first node in the path. Such a sequence is called a

path name. If the first node in a path name is the root of the naming graph, it is called an **absolute path name**. Otherwise, it is called a **relative path name**.

It is important to realize that names are always organized in a name space. As a consequence, a name is always defined relative only to a directory node.

A **global name** is a name that denotes the same entity, no matter where that name is used in a system. In other words, a global name is always interpreted with respect to the same directory node. In contrast, a **local name** is a name whose interpretation depends on where that name is being used. Put differently, a local name is essentially a relative name whose directory in which it is contained is (implicitly) known.

Name resolution

Name spaces offer a convenient mechanism for storing and retrieving information about entities by means of names. More generally, given a path name, it should be possible to look up any information stored in the node referred to by that name. The process of looking up a name is called **name resolution**.

To explain how name resolution works, let us consider a path name such as N:[label1, label2, ..., labeln]. Resolution of this name starts at node N of the naming graph, where the name label1 is looked up in the directory table, and which returns the identifier of the node to which label1 refers. Resolution then continues at the identified node by looking up the name label2 in its directory table, and so on. Assuming that the named path actually exists, resolution stops at the last node referred to by labeln, by returning that node's content.

Closure mechanism: Name resolution can take place only if we know how and where to start. In our example, the starting node was given, and we assumed we had access to its directory table. Knowing how and where to start name resolution is generally referred to as a **closure mechanism**. Essentially, a closure mechanism deals with selecting the initial node in a name space from which name resolution is to start. What makes closure mechanisms sometimes hard to understand is that they are necessarily partly implicit and may be very different when comparing them to each other.

Consider, for example, the string "00312059837784". Many people will not know what to do with these numbers, unless they are told that the sequence is a telephone number. That information is enough to start the resolution process, in particular, by dialing the number. The telephone system subsequently does the rest.

Linking and mounting: Strongly related to name resolution is the use of **aliases**. An alias is another name for the same entity. An environment variable is an example of an alias. In terms of naming graphs, there are basically two different ways to implement an alias.

The first approach is to simply allow multiple absolute paths names to refer to the same node in a naming graph. This approach is illustrated in Figure above, in which node n5 can be referred to by two different path names.

The second approach is to represent an entity by a leaf node, say N, but instead of storing the address or state of that entity, the node stores an absolute path name. When first resolving an absolute path name that leads to N, name resolution will return the path name stored in N, at which point it can continue with resolving that new path name.

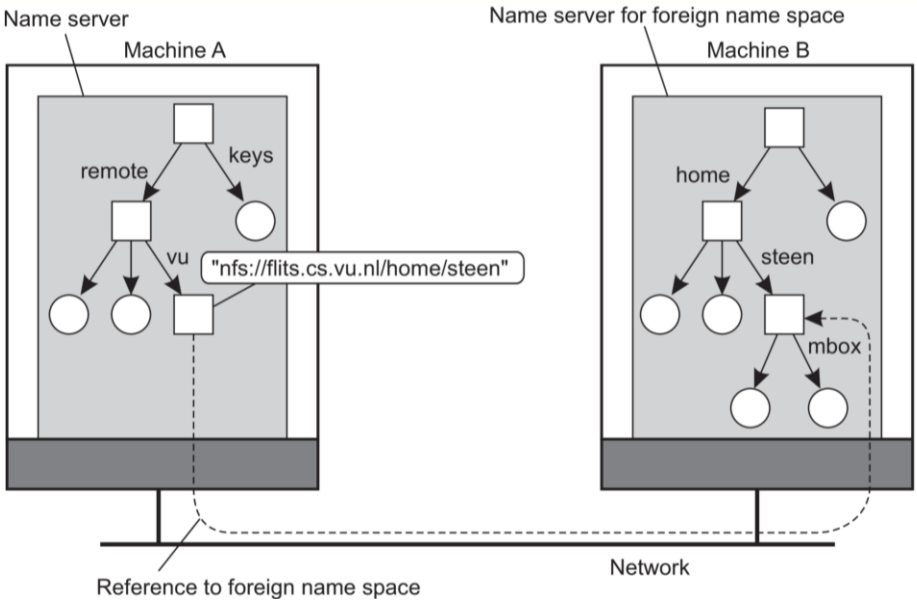
Name resolution as described so far takes place completely within a single name space. However, name resolution can also be used to merge different name spaces in a transparent way. This can be done by mounting file system.

A mounted file system corresponds to letting a directory node store the identifier of a directory node from a different name space, which we refer to as a **foreign name space**. The directory node storing the node identifier is called a **mount point**. Accordingly, the directory node in the foreign name space is called a **mounting point**. Normally, the mounting point is the root of a name space. During name resolution, the mounting point is looked up and resolution proceeds by accessing its directory table.

To mount a foreign name space in a distributed system requires at least the following information:

1. The name of an access protocol.
2. The name of the server.
3. The name of the mounting point in the foreign name space.

Note that each of these names needs to be resolved. The name of an access protocol needs to be resolved to the implementation of a protocol by which communication with the server of the foreign name space can take place. The name of the server needs to be resolved to an address where that server can be reached. As the last part in name resolution, the name of the mounting point needs to be resolved to a node identifier in the foreign name space.



The implementation of a name space

A name space forms the heart of a naming service, that is, a service that allows users and processes to add, remove, and look up names. A naming service is implemented by name servers. If a distributed system is restricted to a local-area network, it is often feasible to implement a naming service by means of only a single name server. However, in large-scale distributed systems with many entities, possibly spread across a large geographical area, it is necessary to distribute the implementation of a name space over multiple name servers.

Name space distribution: Name spaces for a large-scale, possibly worldwide distributed system, are usually organized hierarchically. To effectively implement such a name space, it is convenient to partition it into logical layers, such as:

- The **global layer** is formed by highest-level nodes, that is, the root node and other directory nodes logically close to the root, namely its children. Nodes in the global layer are often characterized by their stability, in the sense that directory tables are rarely changed. Such nodes may represent organizations, or groups of organizations, for which names are stored in the name space.
- The **administrational layer** is formed by directory nodes that together are managed within a single organization. A characteristic feature of the directory nodes in the administrational layer is that they represent groups of entities that belong to the same organization or administrational unit. For example, there may be a directory node for each department in an organization, or a directory node from which all hosts can be found.
- The **managerial layer** consists of nodes that may typically change regularly. For example, nodes representing hosts in the local network belong to this layer. The layer includes nodes representing shared files, user-defined directories and files.

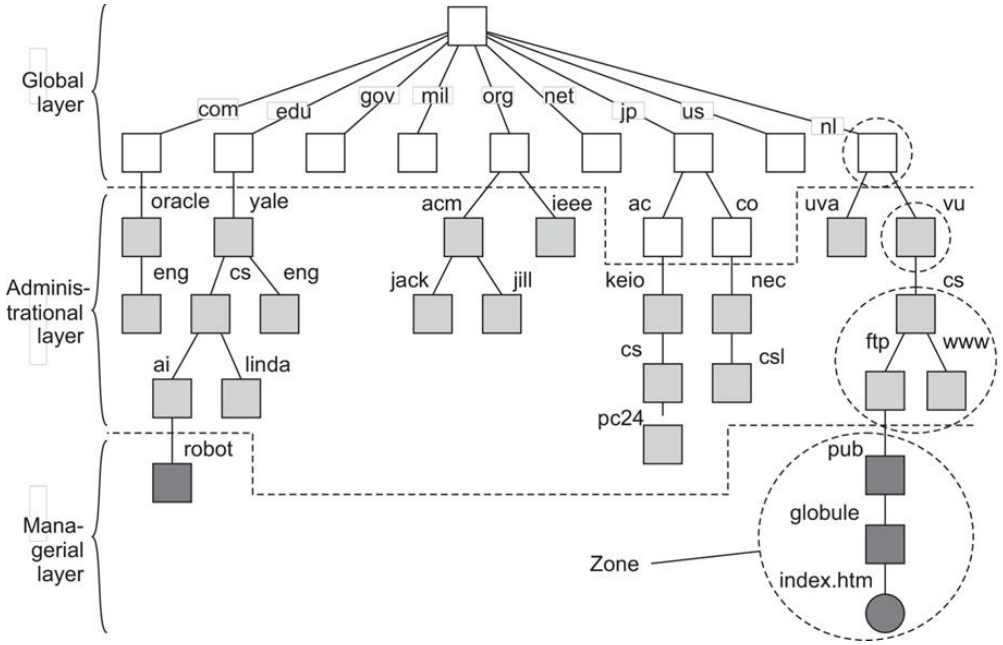


Figure: An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

Table: A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, an administrational layer, and a managerial layer.

Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organizational	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied	Yes	Yes	Sometimes

Implementation of name resolution

The distribution of a name space across multiple name servers affects the implementation of name resolution. To explain the implementation of name resolution in large-scale name services, we assume for the moment that name servers are not replicated and that no client-side caches are used. Each client has access to a local **name resolver**, which is responsible for ensuring that the name resolution process is carried out. There are now two ways to implement name resolution.

1. **Iterative name resolution**: In iterative name resolution, a name resolver hands over the complete name to the root name server. It is assumed that the address where the root server can be contacted is well known. The root server will resolve the path name as far as it can, and return the result to the client. In our example, the root server can resolve only the label nl, for which it will return the address of the associated name server.

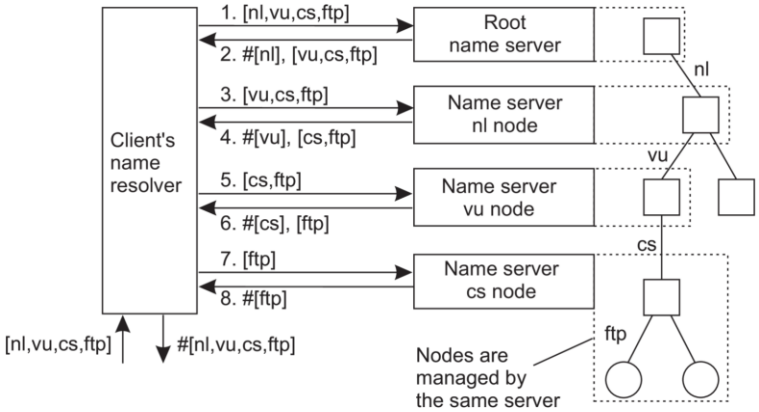


Figure: The principle of iterative name resolution

2. **Recursive name resolution**: An alternative to iterative name resolution is to use recursion during name resolution. Instead of returning each intermediate result back to the client's name resolver, with recursive name resolution, a name server passes the result to the next name server it finds. So, for example, when the root name server finds the address of the name server implementing the node named nl, it requests that name server to resolve the path name nl:[vu, cs, ftp, pub, globe, index.html]. Using recursive name resolution as well, this next server will resolve the complete path and eventually return the file index.html. to the root server, which, in turn, will pass that file to the client's name resolver.

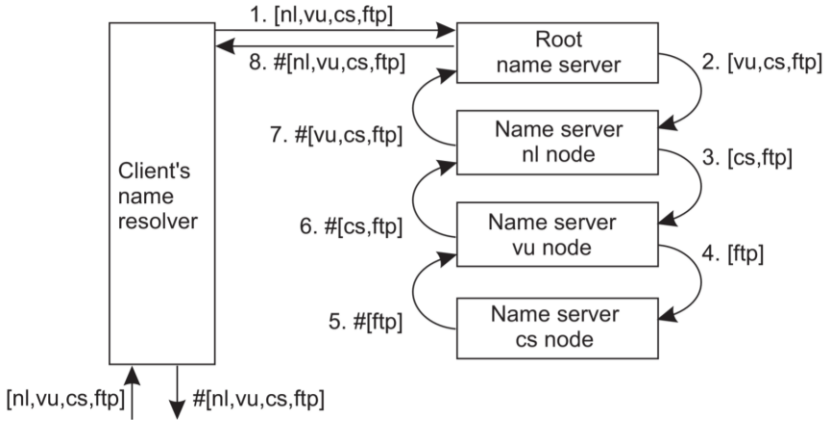


Figure: The principle of recursive name resolution.

The main drawback of recursive name resolution is that it puts a higher performance demand on each name server.

There are two important advantages to recursive name resolution. The first advantage is that caching results is more effective compared to iterative name resolution. The second advantage is that communication costs may be reduced.

Example: The Domain Name System

One of the largest distributed naming services in use today is the Internet Domain Name System (DNS). DNS is primarily used for looking up IP addresses of hosts and mail servers.

The DNS name space

The DNS name space is hierarchically organized as a rooted tree. A label is a case-insensitive string made up of alphanumeric characters. A label has a maximum length of 63 characters; the length of a complete path name is restricted to 255 characters. The string representation of a path name consists of listing its labels, starting with the rightmost one, and separating the labels by a dot (“.”). The root is represented by a dot. So, for example, the path name root:[nl, vu, cs, flits], is represented by the string “flits.cs.vu.nl.”, which includes the rightmost dot to indicate the root node. We generally omit this dot for readability.

Because each node in the DNS name space has exactly one incoming edge (with the exception of the root node, which has no incoming edges), the label attached to a node’s incoming edge is also used as the name for that node. A subtree is called a domain a path name to its root node is called a domain name. Note that, just like a path name, a domain name can be either absolute or relative.

DNS implementation

In essence, the DNS name space can be divided into a global layer and an administrational layer as shown in Figure above. The managerial layer, which is generally formed by local file systems, is formally not part of DNS and is therefore also not managed by it.

5.3 Attribute-based naming

Flat and structured names generally provide a unique and location-independent way of referring to entities. Moreover, structured names have been partly designed to provide a human-friendly way to name entities so that they can be conveniently accessed. In most cases, it is assumed that the name refers to only a single entity.

There are many ways in which descriptions can be provided, but a popular one in distributed systems is to describe an entity in terms of (attribute, value) pairs, generally referred to as **attribute-based naming**. In this approach, an entity is assumed to have an associated collection of attributes. Each attribute says something about that entity. By specifying which values a specific attribute should have, a user essentially constrains the set of entities that he is interested in. It is up to the naming system to return one or more entities that meet the user’s description.

Directory services

Attribute-based naming systems are also known as directory services, whereas systems that support structured naming are generally called naming systems. With directory services, entities have a set of associated attributes that can be used for searching. In some cases, the choice of attributes can be relatively simple. For example, in an e-mail system, messages can be tagged with attributes for the sender, recipient, subject, and so on. However, even in the case of e-mail, matters become difficult when other

types of descriptors are needed, as is illustrated by the difficulty of developing filters that will allow only certain messages (based on their descriptors) to be passed through.

Hierarchical implementations: LDAP

A common approach to tackling distributed directory services is to combine structured naming with attribute-based naming. This approach has been widely adopted, for example, in Microsoft's Active Directory service and other systems. Many of these systems use, or rely on the **lightweight directory access protocol** commonly referred simply as **LDAP**. The LDAP directory service has been derived from OSI's X.500 directory service. As with many OSI services, the quality of their associated implementations hindered widespread use, and simplifications were needed to make it useful.

Conceptually, an LDAP directory service consists of a number of records, usually referred to as directory entries. A directory entry is comparable to a resource record in DNS. Each record is made up of a collection of (attribute, value) pairs, where each attribute has an associated type. A distinction is made between single-valued attributes and multiple-valued attributes. The latter typically represent arrays and lists. As an example, a simple directory entry identifying the network addresses of some general servers from Figure 5.23 is shown in Figure 5.28.

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	VU University
OrganizationalUnit	OU	Computer Science
CommonName	CN	Main server
Mail_Servers	–	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	–	130.37.20.20
WWW_Server	–	130.37.20.20

Figure 5.28: A simple example of an LDAP directory entry using LDAP naming conventions.

5.4 Case Study: The Global Name Service

Task

Practice Questions

1. Explain the terms name, identifiers and addresses.
2. What are the desirable features of a good naming system? Explain.
3. Explain the structured naming scheme of an entity in detail.
4. What is name space? Explain with example.
5. Differentiate between absolute path and relative path in name space.
6. What is name resolution? Explain the different ways to implement name resolution.
7. What do you mean by directory service? Explain.
8. Explain LDAP as hierarchical implementation of attribute-based naming.
9. What is global name service? Explain with its working process.
10. Define the terms closure mechanism, linking and mounting.