Name Service Design in a Multi-Server Operating System

Konstantin Bender, Anton Hergenröder,
Timo Bingmann

June 1, 2006

Roadmap

Goals

Unified Name Space of Objects

Goals: Human Name Space

User and programs can browse and lookup objects.

Consequences

- Names are human readable strings.
- Hierarchical name space
  (humans love to categorize things)
- Performance is important.
  $\longrightarrow$ minimize IPC calls

Goals: Flexibility

Store arbitrary objects in the name space.

We take a look at potential objects in L4:

- threads
- services
- address spaces
- tasks
- files
- others

(the usual suspects)

Goals: Simplicity / Unification

Simple to implement for naming client and naming server.

- We want to use it.
- We want server to be able to easily participate in the name space.
- A client can browse the name space without knowledge of every object type.

Object Representation

Potential objects:

- threads
- services
- address spaces
- tasks
- files
- others

Object Representation

All are identifiable by

object type	possibly an IDL interface
object server	location of the object
object handle	4 byte opaque value

Write as (type, server, handle) tuple.

Fixed length for all objects.

Catalogs

A name gets bound to an object.

ns-slides.pdf $\longrightarrow$ (file_typeid, 42, 512)

Catalogs

A name gets bound to an object.

Group multiple names into a catalog.

ns-slides.pdf $\longrightarrow$ (file_typeid, 42, 512)
ns-slides.tex $\longrightarrow$ (file_typeid, 42, 513)
notes.txt $\longrightarrow$ (file_typeid, 42, 515)

Simple map of strings to objects.

Depth

Create depth by introducing a special object type:

catalog

(think of it as directory)

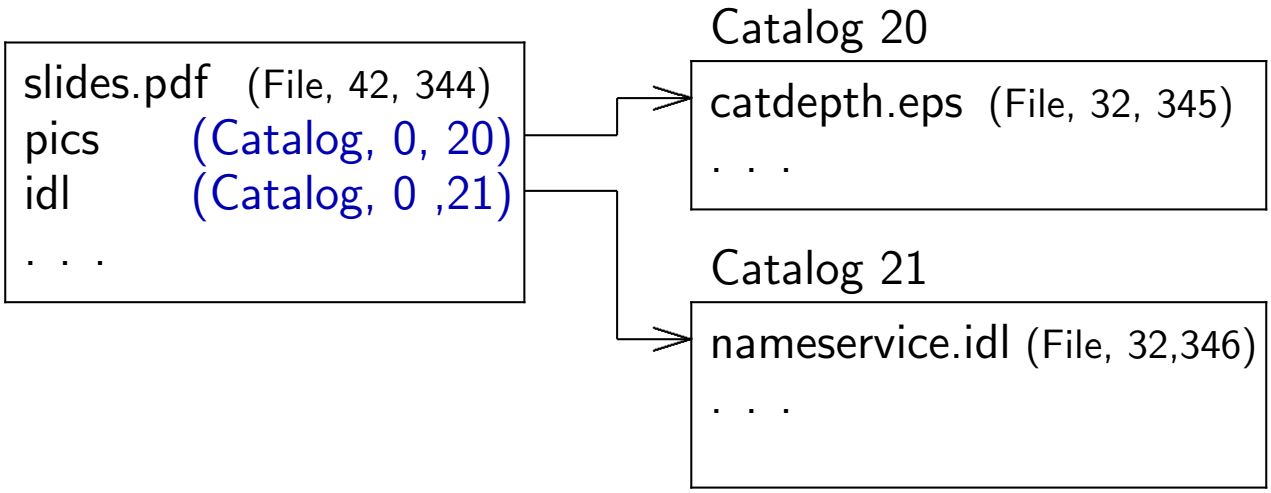
Depth

Create depth by introducing a special object type:

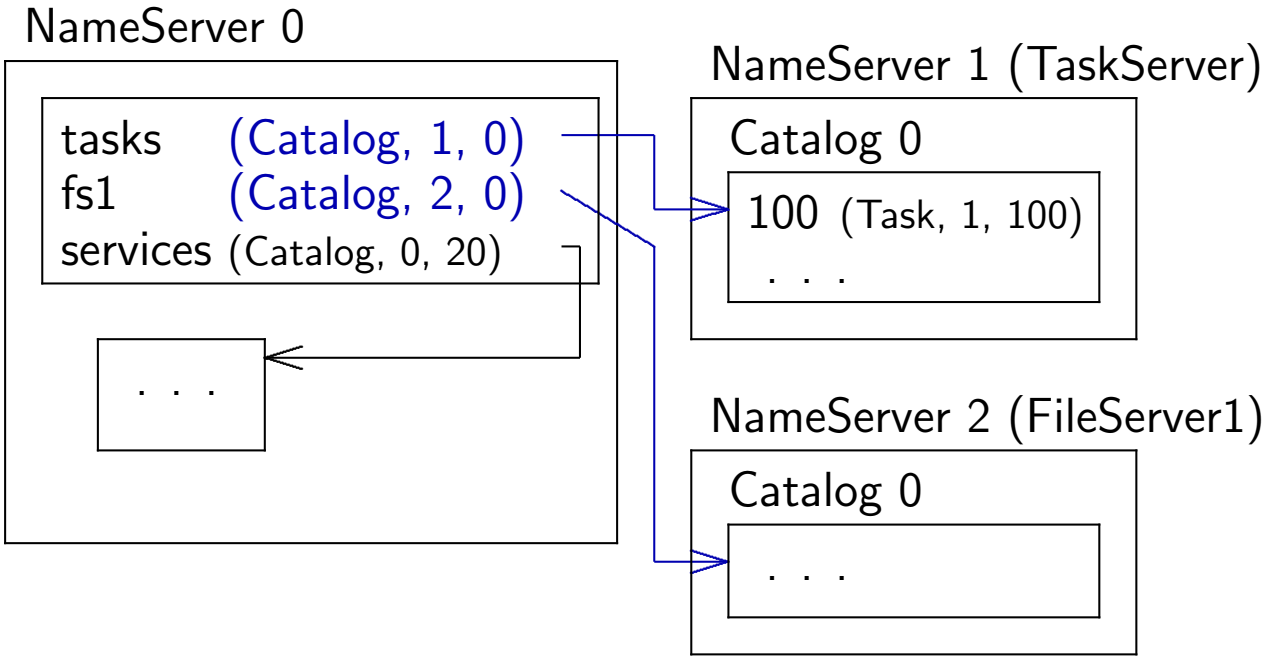
catalog

object type	the name service interface itself
object server	the name server serving the directory
object handle	a catalog id within the server

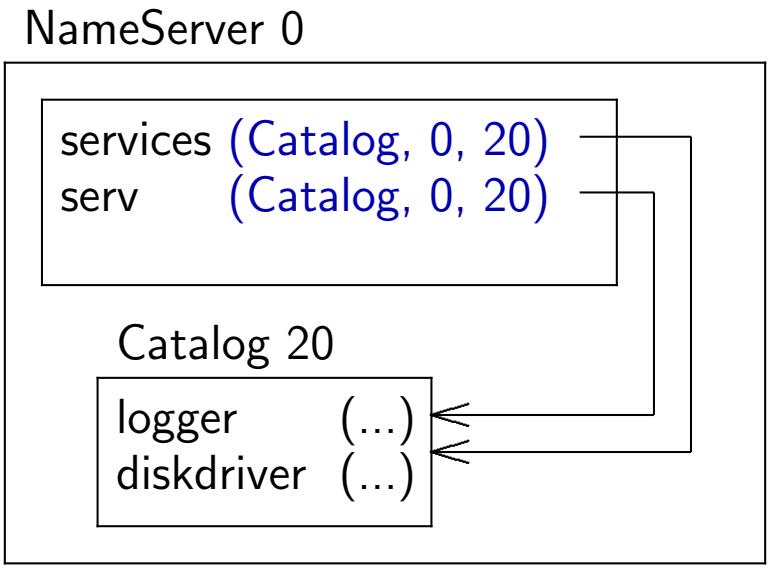
Depth: Subcatalogs



Depth: Mount Points



Depth: Catalog Hard-Links

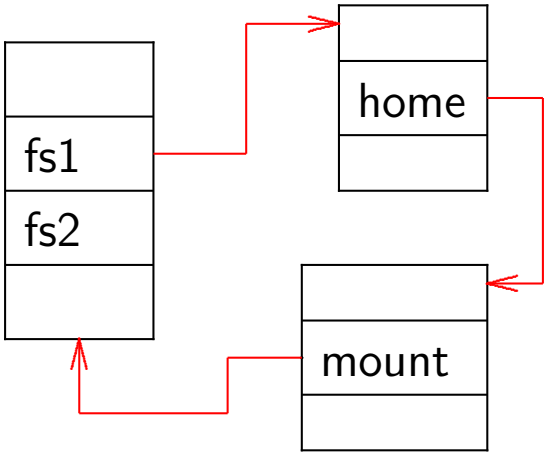


Infinite Depth

Problem

Name space can be a cyclic graph.

Recursive name space walk will run into an infinite loop.



Depth: Closure

Define a Root Name Server.

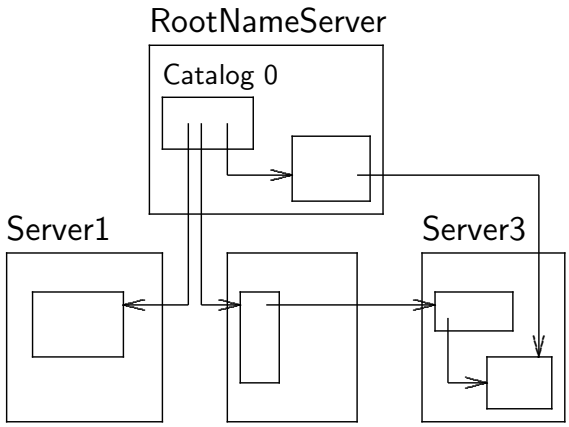
Straight-forward: define fixed thread id.

Implemented as a constant in the name resolve library.

Catalog closure: root catalog on each name server has CatalogId 0.

Root Name Server

The Root Name Server implements the base catalog system.



- Servers can register objects directly.
  $\longrightarrow$ fast single call resolve
- Other name servers can create mount points.
  $\longrightarrow$ distributed autonomous name spaces

IDL Interfaces

We provide two name service interfaces:

Resolve Implemented by all name servers.

Bind Available in the root name server and others.

Resolve Interface

```
module NamingService
{
    struct NameEntry_t
    {
        unsigned long type;
        L4_ThreadId_t server;
        unsigned long handle;
    };

    typedef unsigned long CatalogId_t;

    typedef string StringEntry_t;
    typedef sequence<StringEntry_t> StringList_t;

    typedef sequence<NameEntry_t> NameEntryList_t;
};
```

Resolve Interface

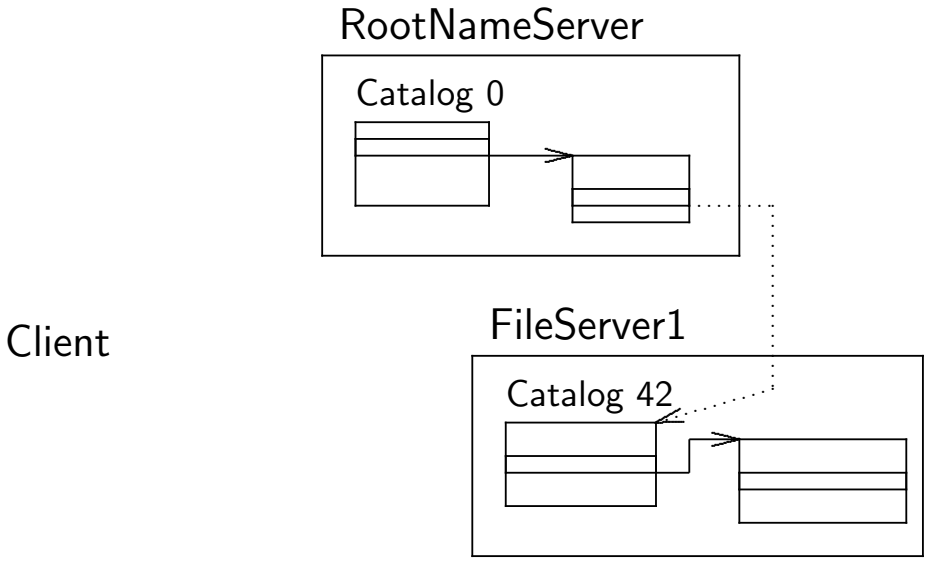
```
module NamingService
{
    interface Resolve
    {
        void Resolve(in CatalogId_t catalogId,
                     in string path,
                     out NameEntry_t entry,
                     out long consumedChars)
            raises(NotFound, InvalidCatalogId);

        void List(in CatalogId_t catalogId,
                  out StringList_t entryNames,
                  out NameEntryList_t entries)
            raises(NotFound, InvalidCatalogId);
    };
};
```


Resolve

```
void Resolve(in CatalogId_t catalogId,
             in string path,
             out NameEntry_t entry,
             out long consumedChars);
```

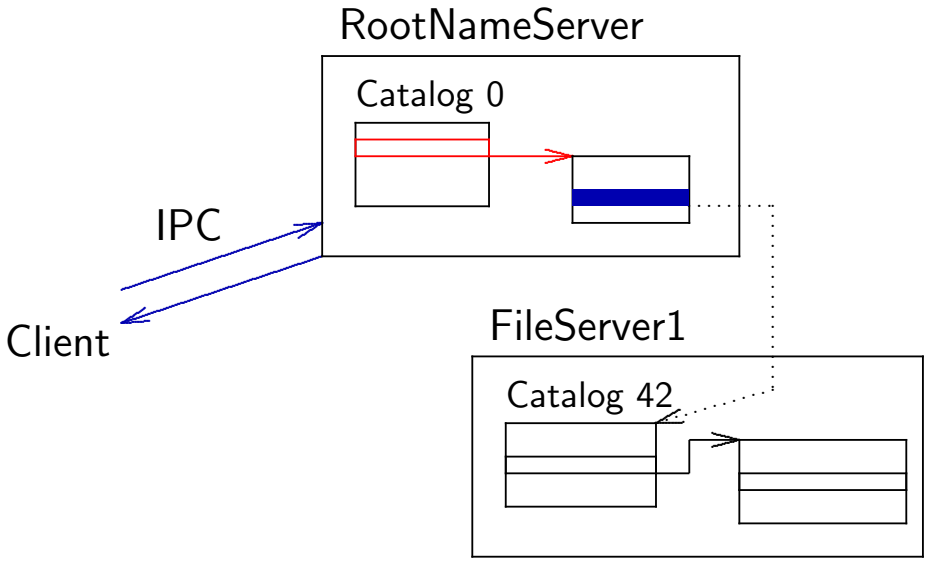
- Resolve starts at catalogId.
- As much of the path is resolved as possible without crossing servers.
- Components of the path are separated by /
- path does not begin with a /
- Client can continue resolve on different server.
- Raises NotFound exception at a dead-end.

Iterative Resolve



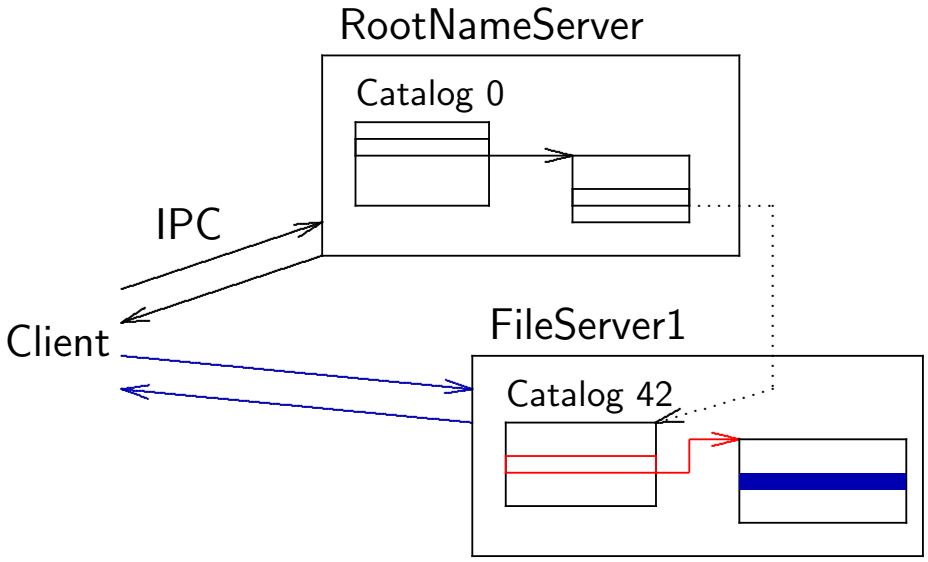
RootNS.Resolve(0, "fs/s1/home/blah")

Iterative Resolve



RootNS.Resolve(0, "fs/s1/home/blah")
= (Catalog, FileServer1, 42) consumed 6

Iterative Resolve



RootNS.Resolve(0, "fs/s1/home/blah")
= (Catalog, FileServer1, 42) consumed 6

FileServer1.Resolve(42, "home/blah")
= (File, FileServer1, 629) consumed 9

List

```
void List(in CatalogId_t catalogId,
          out StringList_t entryNames,
          out NameEntryList_t entries);
```

- Returns names *and* entries of the catalog.
- Used to traverse the name space graph.
- Problem: List can exceed IPC size, `sequence<string>` supported?
- Solution 1: Extend IDL4
- Solution 2: Use `FindFirst` and `FindNext`

Bind Interface

```
module NamingService {
    interface Bind {

        void Bind(in CatalogId_t catalogId,
                  in string path,
                  in NameEntry_t entry)
             raises(NotAllowed, InvalidCatalogId);

        void Unbind(in CatalogId_t catalogId,
                    in string path)
            raises(NotAllowed, NotFound, InvalidCatalogId);

        void Rebind(in CatalogId_t sourceCatalogId,
                    in string sourcePath,
                    in CatalogId_t destinationCatalogId,
                    in string destinationPath)
            raises(NotAllowed, NotFound, InvalidCatalogId);
    };
};
```


Bind Interface

```
void Bind(in CatalogId_t catalogId,
          in string path,
          in NameEntry_t entry);
```

- Registers a new entry in the catalog.
- Automatically creates all non-existing subcatalogs in path.
- The entry.server is considered “owner” of the entry. Only it and the roottask can `unbind` the entry.
- Auto-created subcatalogs are owned by the name server.

Bind Interface

```
void Unbind(in CatalogId_t catalogId,
            in string path);
```

- Removes an entry from the catalog.
- The calling thread must be the owner of the object.
- Path is resolved within the name server.
- All empty subcatalogs except the root are automatically removed.

Bind Interface

```
void Rebind(in CatalogId_t sourceCatalogId,
            in string sourcePath,
            in CatalogId_t destinationCatalogId,
            in string destinationPath)
```

- Atomically changes the name of an entry.
- Paths must be within the same name server.
- Owner access restrictions apply as with `bind` and `unbind`.

Security

- Currently only minimalistic security with `bind`/`unbind` in the Root Name Server.
- First step: split up entry "owner" and entry "maintainer" servers.
- `List` returns all names regardless of access privileges. To fix this a whole user access rights system must be integrated into the name service. Very Difficult.

Symbolic Links

Challenge

- Symbolic Links are absolute paths or relative components within the name space graph.
- They can cross name server boundaries. Catalogs have no parent references → symlinks cannot be implemented in the servers.
- A string cannot be returned using `NameEntry_t`.

Symbolic Links

Possible Solution

- Regard a symlink as an object: handle is an number referencing the link's string.
- Add a required function

  `string readlink(in unsigned long linkid)`

  to the Resolve interface.
- Handle translation of the symlink's string in the name client.

Very Complicated

FindFirst, FindNext

```
module NamingService {
    interface Lookup
    {
        void FindFirst(in CatalogId_t catalogId,
                       out L4_Word_t cookie,
                       out string firstName,
                       out NameEntry_t firstEntry)
            raises(NotFound, InvalidCatalogId);

        void FindNext(in CatalogId_t catalogId,
                      inout L4_Word_t cookie,
                      out string nextName,
                      out NameEntry_t nextEntry)
            raises(NotFound, InvalidCatalogId);
    };
};
```

That's all folks!

Any Questions?