Distributed Operating Systems

What are distributed operating systems?

Optimistic view:

> A distributed system is a collection of independent computers that appear to the users of the system as a single computer. *Tanenbaum*

Pessimistic view:

> A distributed system is where the failure of a computer I have never heard of prevents me from working. *Lamport?*

[The last one is probably slightly misquoted,
I couldn't find the reference - sorry]

Motivation: Advantages of distributed systems

Advantages over centralised systems:

economics: small systems have better price/performance than big ones

speed: collection of small systems can have much more overall power than a large centralised one

reliability: system can survive failure of parts

incremental growth: can add power in small increments

Motivation: Advantages of distributed systems...

Advantages over independent PCs:

data sharing: access of common data base by multiple users

device sharing: common access to expensive peripherals

communication: e-mail etc.

flexibility and utilisation: spread workload (load balancing)

Disadvantages of distributed systems

reliability: increased number of failure points

inadequate software: few real distributed systems exist, those which do exist are not compatible with traditional systems

networking problems: high dependence on network may lead to saturation

security: much more difficult than in other systems to protect data against unauthorised access

Design issues: Transparency

Location transparency: Users don't need to know/cannot tell location of resources

Migration transparency: Resources can move around without changing access

Replication transparency: Multiple copies may exist without users knowing

Concurrency transparency: Users can share resources automatically

Parallelism transparency: Activities can happen in parallel without users knowing

Other design issues

Flexibility: Systems under development, requirements change
⇒ must be able to adapt easily

Reliability: Distributed system should me *more* reliable than others

- higher availability of total system
- fault tolerance: mask failures

Performance: overhead must be less than gain due to replication of hardware

Scalability: system must be able to grow without significant loss of perfromance

Naming in Distributed Systems

- Names required to identify and *locate* objects
- Sharing (of information & services) is one of the main issues in distributed systems.
- Different processes often use different names for the same objects

⇒ *resolution* or *binding* of different names within different processes' contexts yields same result.

Communication between processes about objects requires:

- universally valid names, or
- transmission of context as well as names, or
- ensuring that resolution always happens within the *same* context.

To facilitate sharing, each object (or service) of potential interest to other objects should be given a name which uniquely identifies that object.

Need names for

- computers (mozart, faure, . . .)
- services (mail, printing, . . .)
- ports or other IPC channels
- users
- data objects (e.g. files)

A naming scheme is a mapping from **names** to **objects**.

There are two different approaches to global naming:
unique identifiers (UIDs) and *human-readable names*

UIDs are

- large (≥128-bit) integers
- never re-used
- always referring to the same object (or none)
- location independent (can be capabilities: knowing name implies access)

Names may be *pure*, i.e. "just" bit patterns

- no semantics other than identifying object
- no operations other than ==, *
- imply indirection to locate object

or *impure*

- confer information about the object
  (e.g. location, as in http://www.cse.unsw.edu.au/~cs9242).
- imply a directory structure

⇒ impure names carry commitments (e.g. to the location)

⇒ some operations (moving) require invalidating the name
or installing a potentially costly forwarding mechanism).

In distributed systems there is "more naming" than in centralised systems, because there is more binding:

- machines to addresses
- services to machines

Binding needs to be done eventually, but as late as possible to maintain flexibility.

Bindings (e.g. server location) need to be cached for efficiency (works well for infrequently changing bindings).

Directories should be replicated for efficiency and robustness.

Problem with pure names: where to look up?
⇒ broadcast, hints

Consistency (short term)

Need to incorporate updates.
Strict consistency ⇒ update may fail!

Naming should **always** work. How?

- don't insist on consistency
  e.g. phone book not up-to-date, but "mostly" correct
- many bindings change infrequently (but not all!)
  location of mail server vs. dynamic load balancing
- incorrect binding will be detected (operation fails)

⇒ broadcast, update if inconsistency found

- incorrect binding may not be fatal

⇒ use old version of file, leave forwarding address

Consistency (long term)

Consistency = keeping updates in order

- Conflicting updates may occur in different parts of system

⇒ *essential* inconsistencies!

⇒ requires update protocol (essentially a cache coherency problem)

Scalability

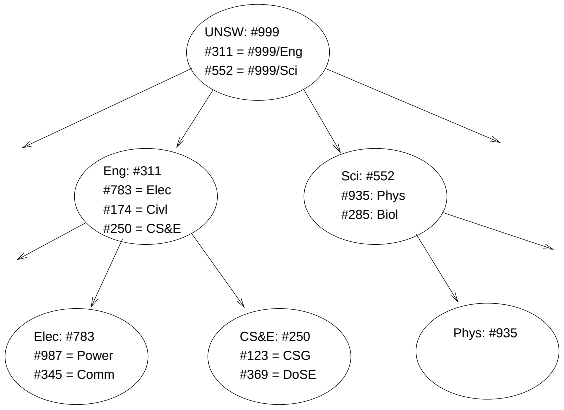
- need to maintain uniqueness of names in growing system
- usual approach: *hierarchical naming*

DEC Global Naming Service (GNS)

GNS [Lampson, 5th SPDC, 1986] was designed to support restructuring of the name space.

- each directory has UID (called "DI")
- hierarchical directory structure (like UNIX)
  files accessible via a path from a root (or CWD)
- alternatively, a file name can descend from a DI
  e.g. if the DI of /a/b/c is #437, the following are equivalent:
  /a/b/c/d/e and #437/d/e

GNS Example



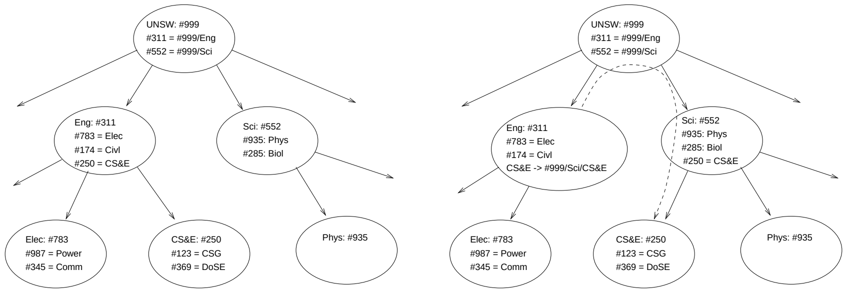
Equivalent names:

/UNSW/Eng/CS&E/CSG/...

#311/CS&E/CSG/...

#250/CSG/...

Restructuring Example



/UNSW/Eng/CS&E/CSG/ #250/CSG/ /UNSW/Sci/CS&E/CSG/

Stanford Design

Stanford Decentralised Global Naming service [Cheriton, Mann, ToCS **7(2), 1989**] completely decouples name and location.

- multi-part object names are *only* naming mechanism (i.e. no UIDs)
- object managers responsible for operations on objects

⇒ object location implies locating manager

- clients caches name bindings (i.e. managers), if obsolete

⇒ broadcast

- inconsistencies are no problem, as obsolete addresses fail

Assume names /a/b/c, /a/b/d, /a/b/e, /a/b/f

- does **not** imply existence of object /a/b

When looking up /a/b/c/d after having previously looked up /a/b

- can treat /a/b as a *context* for c/d and

  use context and c/d to attempt lookup
- contexts are hints only, if fails ⇒ broadcast

Plan 9 from Bell Labs

Plan 9 [Pike, Presotto, Thompson, Trickey, Proc. UKUUG 1990] naming is based on a file system model, in fact, all services on Plan 9 emulate file systems. Accordingly, all names look like file names.

Most striking feature:

- name space is not the same for all users,
  users tailor their name spaces.
- system supports a location-independent view of the name-space *for each user*, although this is not mandatory.

Plan 9 name spaces

- global name space for servers
  e.g. nj/mh/astro/helix
- local name space for each process
  usually all of a user's processes share the same local name space
- calling any service connects to the root of the service's name space
- service's name space is joined to users' by "attach" operation, leads to a union of the name spaces
- options determine how to resolve clashes
  i.e. old or new names take precedence
- communication with service is via files

Examples

OS sets up for each process several directories for accessing standard services

/proc	access to process data
/dev	access to devices
/dev/cons	standard I/O (to/from window)
/dev/mouse	mouse
/dev/bitblt	graphic access to window
/dev/time	clock
/n/<*fileserver*>	name space of *fileserver*'s files.

/proc directory

/proc/<*PID*>/ contains files for examining or manipulating processes (including self)

/proc/<*PID*>/mem	process' memory
/proc/<*PID*>/ctl	file for process control (stopping, restarting etc)
/proc/<*PID*>/text	executable code file (for debuggers etc)
/proc/<*PID*>/status	process status

Executables & Heterogeneity

All executables (system's as well as user's) are in /bin: the system's /bin is attached to the user's. To deal with heterogeneity, there are directories like /mips/bin, /alpha/bin, the appropriate one is attached to /bin.

Executing the "cpu" command spawns a shell on a different CPU, but *in the same name space*. This is achieved by passing a description of the name space to the CPU server, which can then reconstruct the name space for the new process.

