Lecture #2: Distributed Operating Systems: an introduction

Topics for today

- Overview of major issues in distributed operating systems
- Terminology
- Communication models
- Remote procedure calls

These topics are from Chapter 4 in the *Advanced Concepts in OS* text.

What is a distributed system?

- It consists of multiple computers that do not share a memory.
- Each Computer has its own memory and runs its own operating system.
- The computers can communicate with each other through a communication network.
- See Figure 4.1 for the architecture of a distributed system.

Why build a distributed system?

- Microprocessors are getting more and more powerful.
- A distributed system combines (and increases) the computing power of individual computer.
- Some advantages include:
    - Resource sharing
      (but not as easily as if on the same machine)
    - Enhanced performance
      (but 2 machines are not as good as a single machine that is 2 times as fast)
    - Improved reliability & availability
      (but probability of single failure increases, as does difficulty of recovery)
    - Modular expandability
- Distributed OS's have not been economically successful!!!

System models:

- the minicomputer model (several minicomputers with each computer supporting multiple users and providing access to remote resources).
- the workstation model (each user has a workstation, the system provides some common services, such as a distributed file system).
- the processor pool model (the model allocates processor to a user according to the user's needs).

Where is the knowledge of distributed operating systems likely to be useful?

- custom OS's for high performance computer systems
- OS subsystems, like NFS, NIS

- distributed ``middleware'' for large computations
- distributed applications

Issues in Distributed Systems

- the lack of *global knowledge*
- naming
- scalability
- compatibility
- process synchronization (requires global knowledge)
- resource management (requires global knowledge)
- security
- fault tolerance, error recovery

Lack of Global Knowledge

- *Communication delays* are at the core of the problem
- Information may become false before it can be acted upon
- these create some fundamental problems:
    - no global clock -- scheduling based on fifo queue?
    - no global state -- what is the state of a task? What is a correct program?

Naming

- named objects: computers, users, files, printers, services
- namespace must be large
- unique (or at least unambiguous) names are needed
- logical to physical mapping needed
- mapping must be changeable, expandable, reliable, fast

Scalability

- How large is the system designed for?
- How does increasing number of hosts affect overhead?
- broadcasting primitives, directories stored at every computer -- these design options will not work for large systems.

Compatibility

- Binary level: same architecture (object code)
- Execution level: same source code can be compiled and executed (source code).
- Protocol level: only requires all system components to support a common set of protocols.

Process synchronization

- test-and-set instruction won't work.
- Need all new synchronization mechanisms for distributed systems.

Distributed Resource Management

- Data migration: data are brought to the location that needs them.
  - distributed filesystem (file migration)
  - distributed shared memory (page migration)
- Computation migration: the computation migrates to another location.
  - remote procedure call: computation is done at the remote machine.
  - processes migration: processes are transferred to other processors.

Security

- Authetication: guaranteeing that an entity is what it claims to be.
- Authorization: deciding what privileges an entity has and making only those privileges available.

Structuring

- the monolithic kernel: one piece
- the collective kernel structure: a collection of processes
- object oriented: the services provided by the OS are implemented as a set of objects.
- client-server: servers provide the services and clients use the services.

Communication Networks

- WAN and LAN
- traditional operating systems implement the TCP/IP protocol stack: host to network layer, IP layer, transport layer, application layer.
- Most distributed operating systems are not concerned with the lower layer communication primitives.

Communication Models

- message passing
- remote procedure call (RPC)

Message Passing Primitives

- Send (message, destination), Receive (source, buffer)
- buffered *vs*. unbuffered
- blocking *vs*. nonblocking
- reliable *vs*. unreliable
- synchronous *vs*. asynchronous

Example: Unix socket I/O primitives

```
#include <sys/socket.h>
ssize_t sendto(int socket, const void *message,
  size_t length, int flags,
  const struct sockaddr *dest_addr, size_t dest_len);
ssize_t recvfrom(int socket, void *buffer,
  size_t length, int flags, struct sockaddr *address,
```

```
  size_t *address_len);
int poll(struct pollfd fds[], nfds_t nfds,
  int timeout);
int select(int nfds, fd_set *readfds, fd_set *writefds,
  fd_set *errorfds, struct timeval *timeout);
```

You can find more information on these and other socket I/O operations in the Unix `man` pages.

RPC

With message passing, the application programmer must worry about many details:

- parsing messages
- pairing responses with request messages
- converting between data representations
- knowing the address of the remote machine/server
- handling communication and system failures

RPC is introduced to help hide and automate these details.

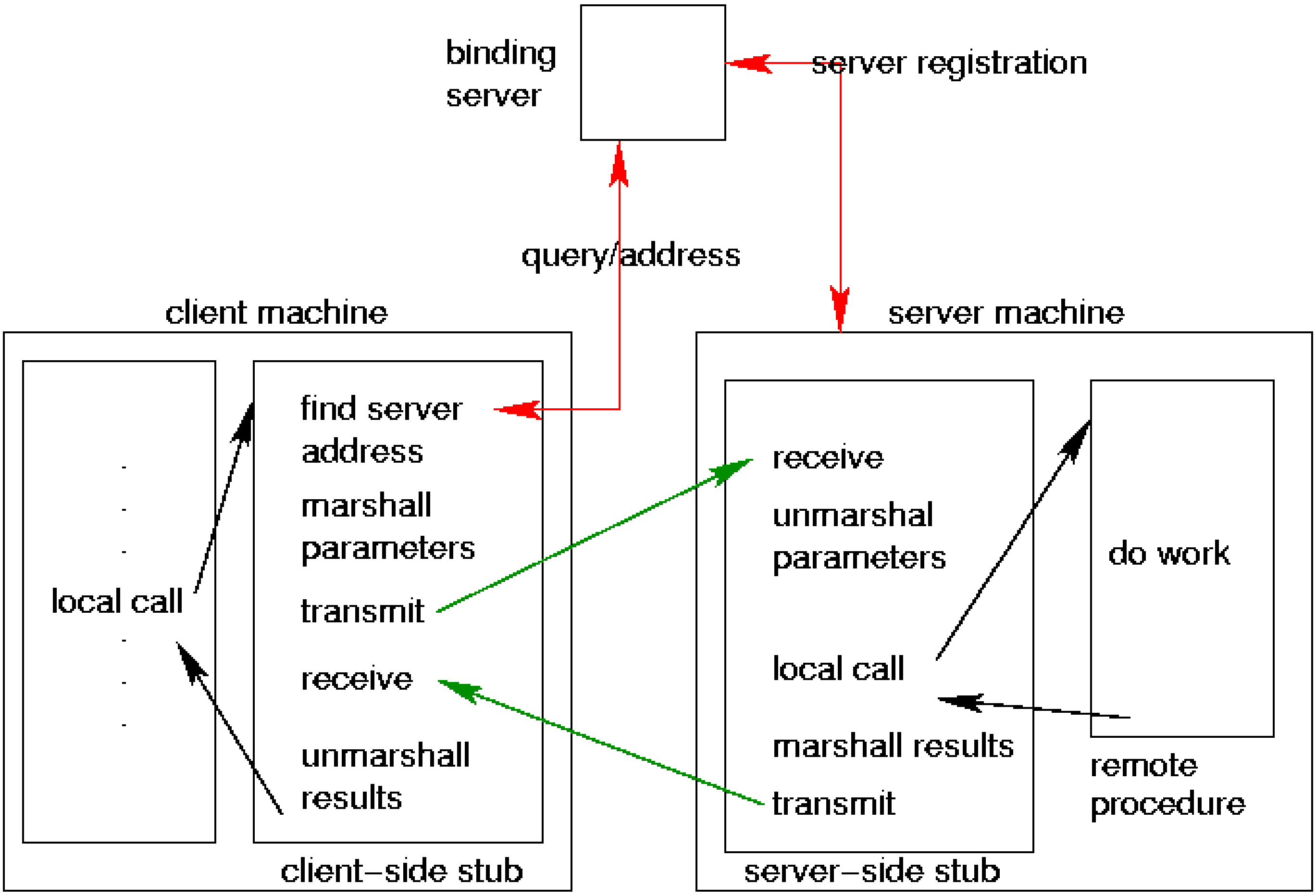
RPC is based on a ``virtual'' procedure call model

- client calls server, specifying operation and arguments
- server executes operation, returning results

RPC Issues

- Stubs (See Unix `rpcgen` tool, for example.)
  - are automatically generated, e.g. by compiler
  - do the ``dirty work'' of communication
- Binding method
  - server address may be looked up by service-name
  - or port number may be looked up
- Parameter and result passing
- Error handling semantics

RPC Diagram


binding server
server registration
query/address
client machine
server machine
find server address
marshall parameters
transmit
receive
unmarshall results
local call
client–side stub
receive
unmarshal parameters
local call
marshall results
transmit
server–side stub
do work
remote procedure