# Client / Server Programming

COMP476
Networked Computer Systems

## Client-Server Paradigm

- *Server* application is “listener”
  - Waits for incoming message
  - Performs service
  - Returns results
- *Client* application establishes connection
  - Sends message to server
  - Waits for return message

## Characteristics of Client

- Arbitrary application program
  - Becomes client when network service is needed
  - Also performs other computations
- Invoked directly by user
- Runs locally on user's computer
- Initiates contact with server
- Can access multiple services

## Characteristics of Server

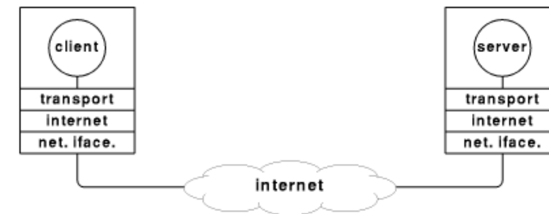
- Special purpose application dedicated to providing network service
- Starts at system initialization time
- Runs on a remote computer (usually centralized, shared computer)
- Waits for service requests from clients; loops to wait for next request
- Will accept requests from arbitrary clients; provides one service to each client

## Message Exchanges

- Typically, client and server exchange messages:
  - Client sends request, perhaps with data
  - Server send response, perhaps with data
- Client may send multiple requests; server sends multiple responses
- Server may send multiple response - consider streaming audio

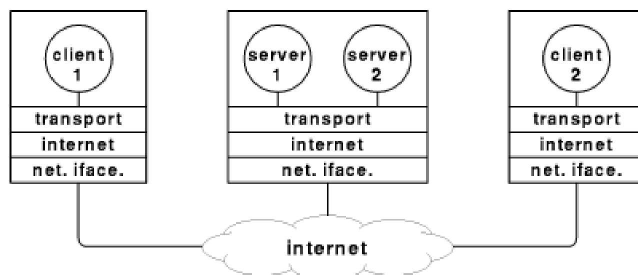
## Transport Protocols and Client-Server Paradigm

- Clients and servers exchange messages through transport protocols; e.g., TCP or UDP
- Both client and server must have same protocol stack and both interact with transport




## Multiple Services on One Computer

- Servers run as independent processes and can manage clients simultaneously




## Multiple Services on One Computer

- Can reduce costs by sharing resources among multiple services
- Reduces management overhead - only one computer to maintain
- One server can affect others by exhausting computer resources
- Failure of single computer can bring down multiple servers

## Selecting from Multiple Servers

- How do incoming messages get delivered to the correct server?
- Each transport session has two unique identifiers
  - (IP address, port number) on server
  - (IP address, port number) on client
- No two clients on one computer can use same source port
- Thus, client endpoints are unique, and server computer protocol software can deliver messages to correct server process

## Identifying a Service

- Each service gets a *unique identifier*; both client and server use that identifier
  - Server registers with local protocol software under the identifier
  - Client contacts protocol software for session under that identifier
- Example - TCP uses *protocol port numbers* as identifiers
  - Server registers under port number for service
  - Client requests session with port number for service

## Binding of Server Location

- How and when does a client application learn the location of a service?
- Goals:
  - portability - allow the application to be used on different systems
  - load balancing - select server with lowest utilization
  - failure recovery - select a different server if original fails
  - efficiency – avoid many messages or broadcasts

## Finding a Service

- Write server name in code
- Read a file of server addresses
- Broadcast request for a server
- Ask a human
- Name server

## UDP or TCP

- TCP - connection-oriented
  - Client establishes connection to server
  - Client and server exchange multiple messages of arbitrary size
  - Client terminates connection
- UDP - connectionless
  - Client constructs message and sends it to the server
  - Server responds
  - Message must fit in one UDP datagram
- Some services use both

## Client-Server Interactions

- Clients can access multiple services
- Clients may access different servers for one service
- Servers may become clients of other servers

## Client-Server Summary

- *Client-server* paradigm used in almost every distributed computation
  - *Client* requests service when needed
  - *Server* waits for client requests
- Clients and servers use *transport protocols* to communicate
- Often, but not always, there is an *application protocol*

## Remote Function Execution

- When a program calls a function or method (or procedure or subroutine), the function is usually executed as part of the main program.

```
x = func(z);
```

- func is run on the same computer as the main.
- Intermediate software or middleware can be used to execute the procedure on a remote computer.

## Remote Procedure Call (RPC)

- A remote procedure call is a paradigm for writing distributed programs or programs that communication between machines.
- An RPC calls a function or procedure that is executed on another computer.
- RPC's provide a more organized, high level interface to writing distributed software than TCP send and receives.

## RPC Advantages

- Appears to the user like a call to a function on the local machine.
- RPC concept is similar to conventional programming.
- Procedure calling has well understood semantics.
- RPC simplifies access to remote systems.

## Reasons for Remote Procedures

- Data may be centrally located.
- Resource sharing of peripherals.
- Execution of a function may require special hardware (i.e. vector processor).

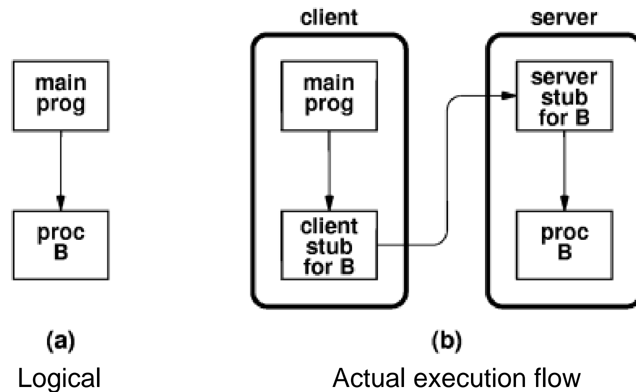
## RPC Construction

- Assume the main program is calling

```
int myfunc(float x);
```

- Write a "*client stub*" called `myfunc` that will not perform the computation, but will send the parameter to a remote system.
- Write a "*server stub*" that will receive the data and call the actual `myfunc` function. The server stub will send back the results.

## RPC Overview




## RPC Operation

- The main program calls the client stub.
- The client stub copies or marshals the parameters into a communication's packet.
- The client stub send an RPC request to the server.
- The server calls the desired function passing the parameters from the communication's packet.
- When the function returns, the server sends the results to the client stub.
- The client stub returns the results to the caller.

## RPC Input Parameters

- Input is data passed from caller to function.
- **basic datatype** - pass by value semantics, copy data to server
- **arrays or structures** - copy data to server, may be a lot of data
- **pointers to single datatype** - copy data item to server
- **pointers in general** - not allowed

## RPC Output Parameters

- Output parameters are passed from the function back to the calling program.
- C programs use pointers to output variables

  example: `x = func(&z);`
- Data, not addresses, copied from server
- Return value copied from server

## RPC Execution Environment

- No global variables
- No environment variables
- No access to files on the client computer.

## Exception Handling

- What does the communication stub do if the first effort to send a message fails?
  - **no retry** RPC may not work
  - **at-least-once** Keep sending until it gets there (*idempotent* functions)
  - **at-most-once** retry but server must filter repeats

## Synchronization

- Normal RPC are synchronous, the calling program waits until the called function completes.
- Normal RPCs do not provide parallelism.
- The logical thread of execution moves to another computer and then back.
- Asynchronous RPCs execute the called function in parallel with the main program.

## Asynchronous RPC

- Call without reply - stub returns to caller after sending message.
- Useful for "void" functions that do not return a result.
- A "print" function might be a good candidate for an asynchronous RPC.

## Call Backs

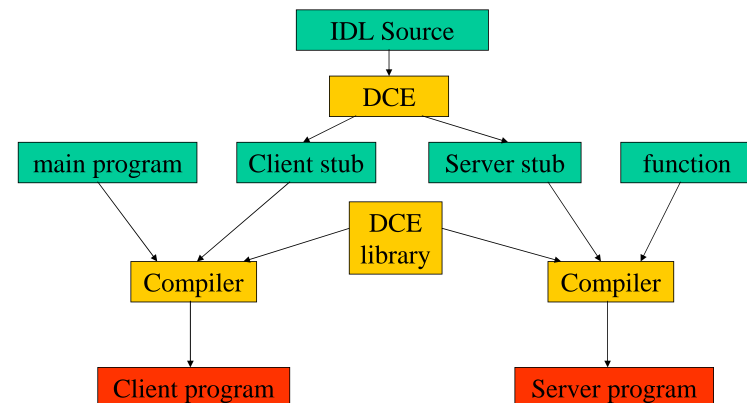
- The RPC returns before the result is complete.
- The server calls a completion function in the client program when the RPC has completed.
- Call backs support event driven programming.

## Automatic Stub Generation

- There are several systems that will automatically generate the client and server stubs.
- Distributed Computing Environment (DCE) from the Open Software Foundation (OSF) supports an RPC system.
- The Interface Definition Language (IDL) defines the function interface similar to a function prototype.

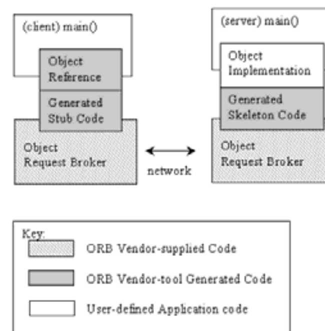
## IDL Example

```
int funcabc(
        [in]        float   x,
        [out]       long   *y,
        [in out]    double *z
        );
```



## Remote Method Invocation

- Object oriented version of RPC
- Supported by Java, CORBA and .NET




## gethostbyname Example

- The gethostbyname(ipname) socket function looks in a database for the given IP name and returns the associated IP address.
- The actual database lookup is done on the Domain Name Server (DNS).

## X- Windows Example

- Main computer is the client. User programs call the server to display data.
- The terminal is the display server.
- Uses Asynchronous RPC.