Distributed Program Design

- Communication-Oriented Design
  - Design protocol first.
  - Build programs that adhere to the protocol.

Typical Sockets Approach

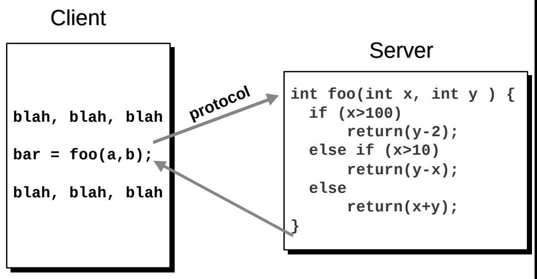
- Application-Oriented Design
  - Build application(s).
  - Divide programs up and add communication protocols.

RPC

RPC
Remote Procedure Call

- Call a procedure (subroutine) that is running on another machine.
- Issues:
  - identifying and accessing the remote procedure
  - parameters
  - return value

Remote Subroutine



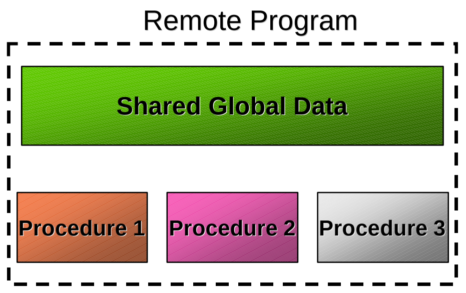
```
blah, blah, blah

bar = foo(a,b);

blah, blah, blah
```

```
int foo(int x, int y ) {
  if (x>100)
      return(y-2);
  else if (x>10)
      return(y-x);
  else
      return(x+y);
}
```

Sun RPC

- There are a number of popular RPC specifications.
- Sun RPC (ONC RPC) is widely used.
- NFS (Network File System) is RPC based.
- Rich set of support tools.

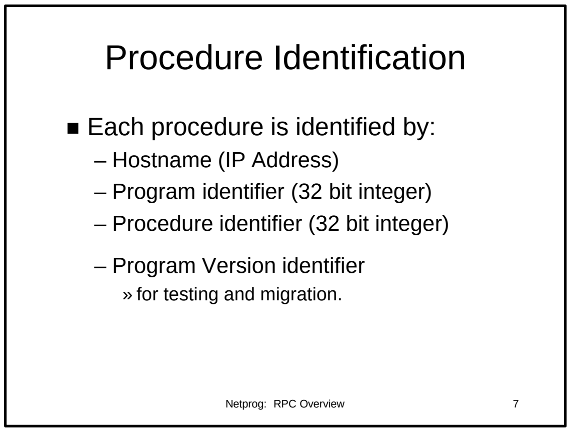
Sun RPC Organization



Procedure Arguments

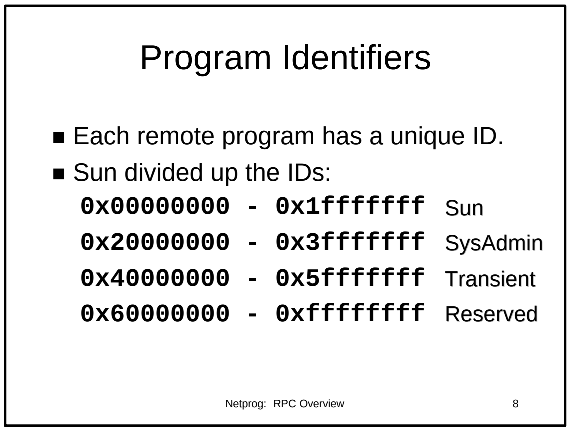
- To reduce the complexity of the interface specification, Sun RPC includes support for a single argument to a remote procedure.*
- Typically the single argument is a structure that contains a number of values.

* Newer versions can handle multiple args.



Procedure Identification

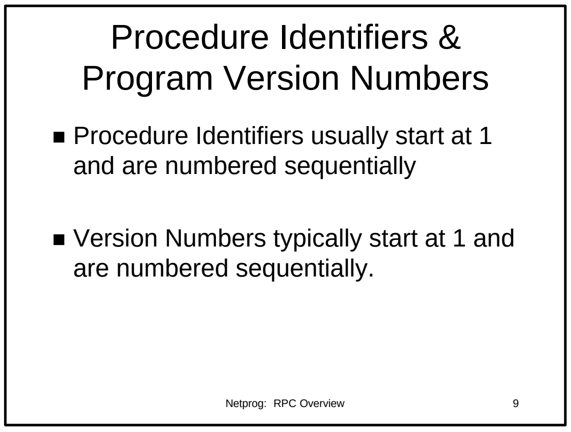
- Each procedure is identified by:
  - – Hostname (IP Address)
  - – Program identifier (32 bit integer)
  - – Procedure identifier (32 bit integer)
  - – Program Version identifier
    - » for testing and migration.



Program Identifiers

- Each remote program has a unique ID.
- Sun divided up the IDs:

```
0x00000000 - 0x1fffffff  Sun
0x20000000 - 0x3fffffff  SysAdmin
0x40000000 - 0x5fffffff  Transient
0x60000000 - 0xffffffff  Reserved
```



Procedure Identifiers & Program Version Numbers

- Procedure Identifiers usually start at 1 and are numbered sequentially
- Version Numbers typically start at 1 and are numbered sequentially.

Iterative Server

- Sun RPC specifies that at *most* one remote procedure within a program can be invoked at any given time.
- If a 2nd procedure is called, the call blocks until the 1st procedure has completed.

Iterative can be good

- Having an iterative server is useful for applications that may share data among procedures.
- Example: database - to avoid insert/delete/modify collisions.
- We can provide concurrency when necessary...

Call Semantics

- What does it mean to call a local procedure?
  - the procedure is run exactly one time.
- What does it mean to call a remote procedure?
  - It might not mean "run exactly once"!

Remote Call Semantics

- To act like a local procedure (exactly one invocation per call) - a reliable transport (TCP) is necessary.
- Sun RPC does not support reliable call semantics. !
- "At Least Once" Semantics
- "Zero or More" Semantics

Sun RPC Call Semantics

- At Least Once Semantics
  – if we get a response (a return value)
- Zero or More Semantics
  – if we don't hear back from the remote subroutine.

Remote Procedure deposit()

```
deposit(DavesAccount,$100)
```

- Always remember that you don't *know* how many times the remote procedure was run!
  – The net can duplicate the request (UDP).

Network Communication

- The actual network communication is nothing new - it's just TCP/IP.
- Many RPC implementations are built upon the sockets library.
  - the RPC library does all the work!
- We are just using a different API, the underlying stuff is the same!

Dynamic Port Mapping

- Servers typically do not use well known protocol ports!
- Clients know the Program ID (and host IP address).
- RPC includes support for looking up the port number of a *remote program.*

Port Lookup Service

- A port lookup service runs on each host that contains RPC servers.
- RPC servers register themselves with this service:
  - "I'm program 17 and I'm looking for requests on port 1736"

The portmapper

- Each system which will support RPC servers runs a *port mapper* server that provides a central registry for RPC services.
- Servers tell the port mapper what services they offer.

More on the portmapper

- Clients ask a remote port mapper for the port number corresponding to Remote Program ID.
- The portmapper is itself an RPC server!
- The portmapper is available on a well-known port (111).

Sun RPC Programming

- The RPC library is a collection of tools for automating the creation of RPC clients and servers.
- RPC clients are processes that call remote procedures.
- RPC servers are processes that include procedure(s) that can be called by clients.

RPC Programming

- RPC library
  - XDR routines
  - RPC run time library
    - call rpc service
    - register with portmapper
    - dispatch incoming request to correct procedure
  - Program Generator

RPC Run-time Library

- High- and Low-level functions that can be used by clients and servers.
- High-level functions provide simple access to RPC services.

High-level Client Library

```
int callrpc( char *host,
             u_long prognum,
             u_long versnum,
             u_long procnum,
             xdrproc_t inproc,
             char *in,
             xdrproc_t outproc,
             char *out);
```

High-Level Server Library

```
int registerrpc(
             u_long prognum,
             u_long versnum,
             u_long procnum,
             char *(*procname)()
             xdrproc_t inproc,
             xdrproc_t outproc);
```

Netprog: RPC Overview 25

High-Level Server Library (cont.)

```
void svc_run();
```

- **`svc_run()`** is a *dispatcher.*
- A dispatcher waits for incoming connections and invokes the appropriate function to handle each incoming request.

Netprog: RPC Overview 26

High-Level Library Limitation

- The High-Level RPC library calls support UDP only (no TCP).
- You must use lower-level RPC library functions to use TCP.
- The High-Level library calls do not support any kind of authentication.

Netprog: RPC Overview 27

Low-level RPC Library

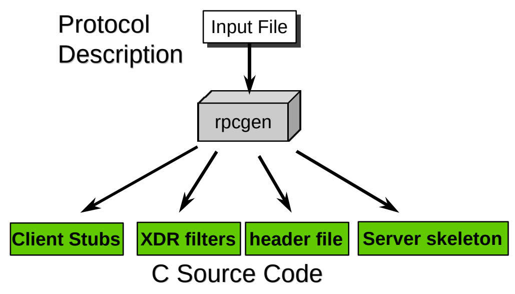
- Full control over all IPC options
  - TCP & UDP
  - Timeout values
  - Asynchronous procedure calls
- Multi-tasking Servers
- Broadcasting

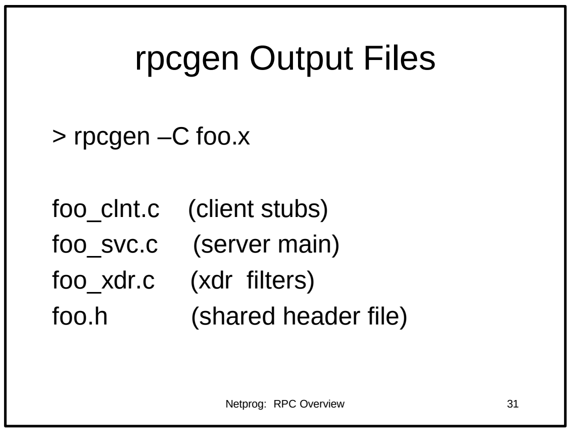
IPC is InterProcess Communication

RPCGEN

- There is a tool for automating the creation of RPC clients and servers.
- The program *rpcgen* does most of the work for you.
- The input to rpcgen is a *protocol definition* in the form of a list of remote procedures and parameter types.

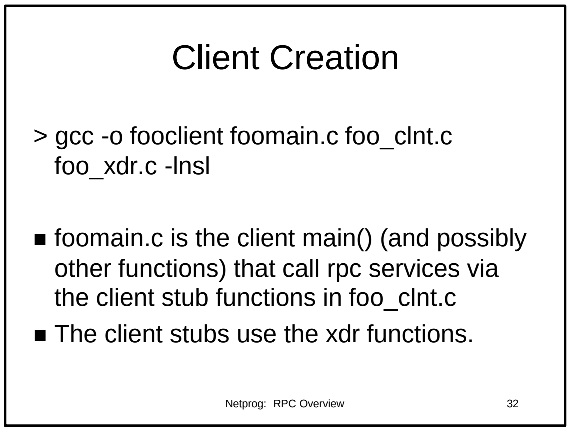
RPCGEN





rpcgen Output Files

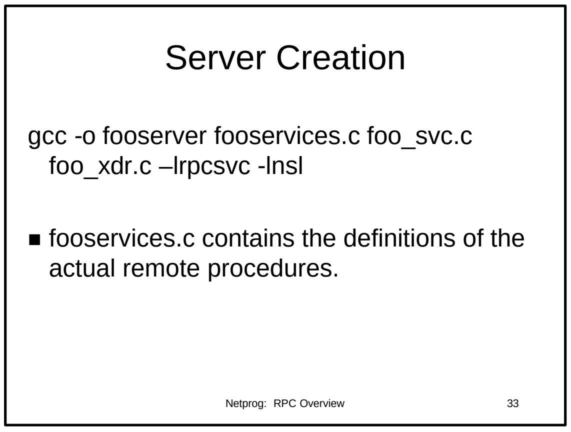
```
> rpcgen –C foo.x

foo_clnt.c    (client stubs)
foo_svc.c     (server main)
foo_xdr.c     (xdr  filters)
foo.h         (shared header file)
```



Client Creation

```
> gcc -o fooclient foomain.c foo_clnt.c
  foo_xdr.c -lnsl
```

- foomain.c is the client main() (and possibly other functions) that call rpc services via the client stub functions in foo_clnt.c
- The client stubs use the xdr functions.



Server Creation

```
gcc -o fooserver fooservices.c foo_svc.c
  foo_xdr.c –lrpcsvc -lnsl
```

- fooservices.c contains the definitions of the actual remote procedures.

Example Protocol Definition

```
struct twonums {
  int a;
  int b;
};
program UIDPROG {
  version UIDVERS {
     int  RGETUID(string<20>) = 1;
     string RGETLOGIN( int ) = 2;
     int  RADD(twonums)  = 3;
  } = 1;
} = 0x20000001;
```