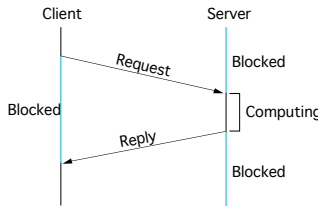
# Remote Procedure Call

Outline
- Concept of RPC
- SunRPC



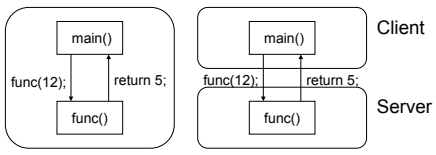
# RPC Timeline



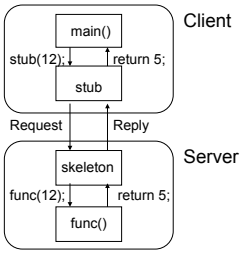



# RPC

- There exists a way for processes to represent a task: procedure/function.



## RPC






## Stubs

- Stub is a function with the same interface as func(); it converts the function call into a network response and a network response into a function return.
- Skeleton converts requests into function calls and function returns into network replies.
- RPC System: used to generate both stub and skeleton (automatically).

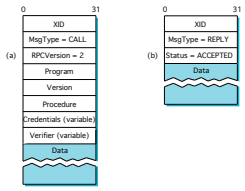


## SunRPC

- Widely used Remote Procedure Call system:
  - Sun Microsystems, implemented in most Unix systems
  - NFS distributed file system is based on Sun RPC
- Designed to call remote C procedures.
- Platform-independent.

## SunRPC Header Format

- XID (transaction id)
- Server does not remember last XID it serviced
- Problem if client retransmits request while reply is in transit






## RPC

- One computer can be server for multiple procedures:
  - server may host several programs (identified by program number)
  - each program may have several subsequent versions (version number)
  - each version may contain one or more procedures (procedure number)
- Program numbers are 32-bit hexadecimal values (0x21212100)
  - As user, you can choose any program number between 0x20000000 and 0x3FFFFFFF, but they have to be unique (not several programs with same number on same machine)
- Version and procedure numbers are integers (1,2,...)

(prog, vers, proc)



## Sun RPC Program Numbers

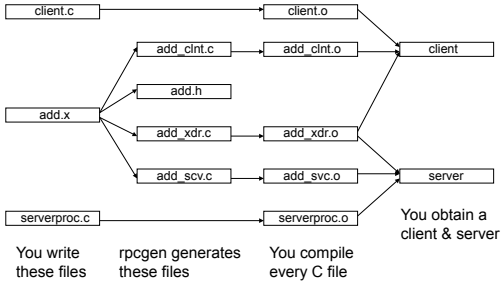
| FROM | TO | VALUES ASSIGNED BY |
|---|---|---|
| 0x00000000 | 0x1fffffff | Sun Microsystems, Inc. |
| 0x20000000 | 0x3fffffff | System manager at a user's site |
| 0x40000000 | 0x5fffffff | Transient (temporary) |
| 0x60000000 | 0x7fffffff | Reserved |
| 0x80000000 | 0x9fffffff | Reserved |
| 0xa0000000 | 0xbfffffff | Reserved |
| 0xc0000000 | 0xdfffffff | Reserved |
| 0xe0000000 | 0xffffffff | Reserved |

## Sun RPC Program Numbers

| NAME | NUMBER | DESCRIPTION |
|---|---|---|
| portmap | 100000 | port mapper |
| rstatd | 100001 | rstat, rup, perfmeter |
| rusersd | 100002 | remote users |
| nfs | 100003 | network file system |
| ypserv | 100004 | yp (NIS) |
| mountd | 100005 | mount, showmount |
| dbxd | 100006 | DMXprog (debugger) |
| ypbind | 100007 | NIS binder |
| … | | |



## Writing an RPC Program






## Example

- Step 1: Write a specification file (add.x)

```
struct add_in { /* Arguments of procedure */
    long arg1;
    long arg2;
};

typedef long add_out; /* Return value */

program ADD_PROG {
    version ADD_VERS {
        add_out ADD_PROC(add_in) = 1 /* Procedure# = 1 */
    } = 1; /* Version# = 1 */
} = 0x3434000; /* Program# = 0x3434000 */
```

## Example

- Contains specifications of:
  - 'add_in' (arguments)
  - typedef 'add_out' (return values)
  - program 'ADD_PROG' (0x3434000)
  - 1 version 'ADD_VERS' (1)
  - 1 procedure 'ADD_PROC' (1)
- Your procedures can only take ONE input argument and return ONE output return value (use structures for more):
  - more readable (structure entries should have meaningful names)



## Example

- Step 2: generate stubs

  *rpcgen add.x*
- 'add.h' contains declarations:

  *#define ADD_PROG 0x34340000 /* Program nb */*
  *#define ADD_VERS 1 /* Version nb */*
  *#define ADD_PROC 1 /* Procedure nb */*
  *add_out * add_proc_1 (add_in *, CLIENT *);*
  *add_out * add_proc_1_svc (add_in *, struct svc_req *);*
  - add_proc_1 is the stub (called by client)
  - add_proc_1_svc is the actual procedure that you will write and run at the server



## Example

- 'add_clnt.c': implementation of 'add_proc_1'
- 'add_svc.c': contains program which calls your procedure 'add_proc_1_svc' when request received
- 'add_xdr.c': marshall/unmarshall routines

## Example

- Step 3: write server procedure: 'serverproc.c'

```
#include "add.h"
add_out *add_proc_1_svc(add_in *in, struct svc_req *rqstp) {
    static add_out out;
    out = in->arg1 + in->arg2;
    return(&out);
}
```

  - rqstp contains some information about the requester (IP address, etc.)



## Example

- Step 4: compile the server
- Need to compile together your procedure, the (generated) server program, the (generated) marshall/unmarshall procedures and the nsl library (contains the RPC runtime).

```
$ gcc -c serverproc.c
$ gcc -c add_svc.c
$ gcc -c add_xdr.c
$ gcc -o server serverproc.o add_svc.o add_xdr.o -lnsl
```

- To start the server:

  ./server



## Example

- Step 5: write a client program client.c

```
#include "add.h"
int main(int argc, char **argv) {
    CLIENT *cl;
    add_in in;
    add_out *out;

    if (argc!=4) {printf("Usage:...\n"); return 1;}

    cl = clnt_create(argv[1], ADD_PROG, ADD_VERS, "tcp");
    in.arg1 = atol(argv[2]);
    in.arg2 = atol(argv[3]);
    out = add_proc_1(&in, cl);
    if (out == NULL) {printf("Error: %s\n", clnt_sperror(cl, argv[1])); }
    else { printf("We received the result: %ld\n", * out); }
    clnt_destroy(cl);
    return 0;
}
```

## Example

- Create a client structure with clnt_create
  *#include <rpc/rpc.h>*
  *CLIENT *clnt_create(char *host, u_long prog, u_long vers, char *proto);*
  - host: name of server machine
  - prog, vers: program/version number
  - proto: transport protocol ("tcp" or "udp")
- You can call add_proc_1 to send the RPC
- When finished, destroy client structure (client structure can be used multiple times without being destroyed and recreated).



## Example

- Step 6: compile the client
  *$ gcc -c client.c*
  *$ gcc -c add_clnt.c*
  *$ gcc -c add_xdr.c*
  *$ gcc -o client client.o add_clnt.o add_xdr.o -lnsl*



## Example

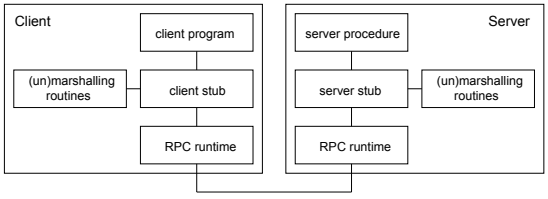
- Step 7: try it out
  - start your server: *./server*
  - send a request:
    *./client machine.cse.nd.edu 8 34*
    *We received the result: 42*

## Mutual Exclusion

- Sun RPC: at most one remote procedure in a remote program can be invoked at a given time.
- Automatic mutual exclusion among procedures within a given remote program.
- No synchronization needed.
- Some versions of rpcgen allow one to generate server code which implement one-thread-per-client (Solaris does, Linux doesn't).



## Inside SunRPC

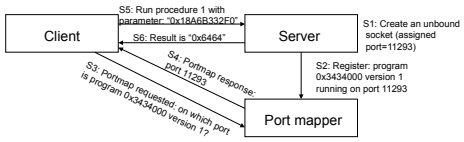





## Low-Level RPC: Port Mapper

- Did you notice that we did not specify a port number for our 'add' server?
  - Could be done with well-known port number for RPCs
  - But then we could not run multiple servers in the same machine simultaneously
- RPC: port mapper
  - server running on port 111
  - when your server starts, it does not bind its socket to a specific port; instead it 'registers' whatever port number it has been given by the system to the local port mapper
  - when you create a client with 'clnt_create', you automatically contact the remote port mapper to ask for the port number of the server you want to contact

# Port Mapper




You can request the port mapper by hand:

*$ rpcinfo -p wizard.cse.nd.edu*

| *program* | *vers* | *proto* | *port* | |
|---|---|---|---|---|
| *100000* | *2* | *tcp* | *111* | *portmapper* |
| *100000* | *2* | *udp* | *111* | *portmapper* |
| *66443998* | *1* | *udp* | *32877* | |
| *66443938* | *1* | *tcp* | *54211* | |