BITS Pilani
Hyderabad Campus

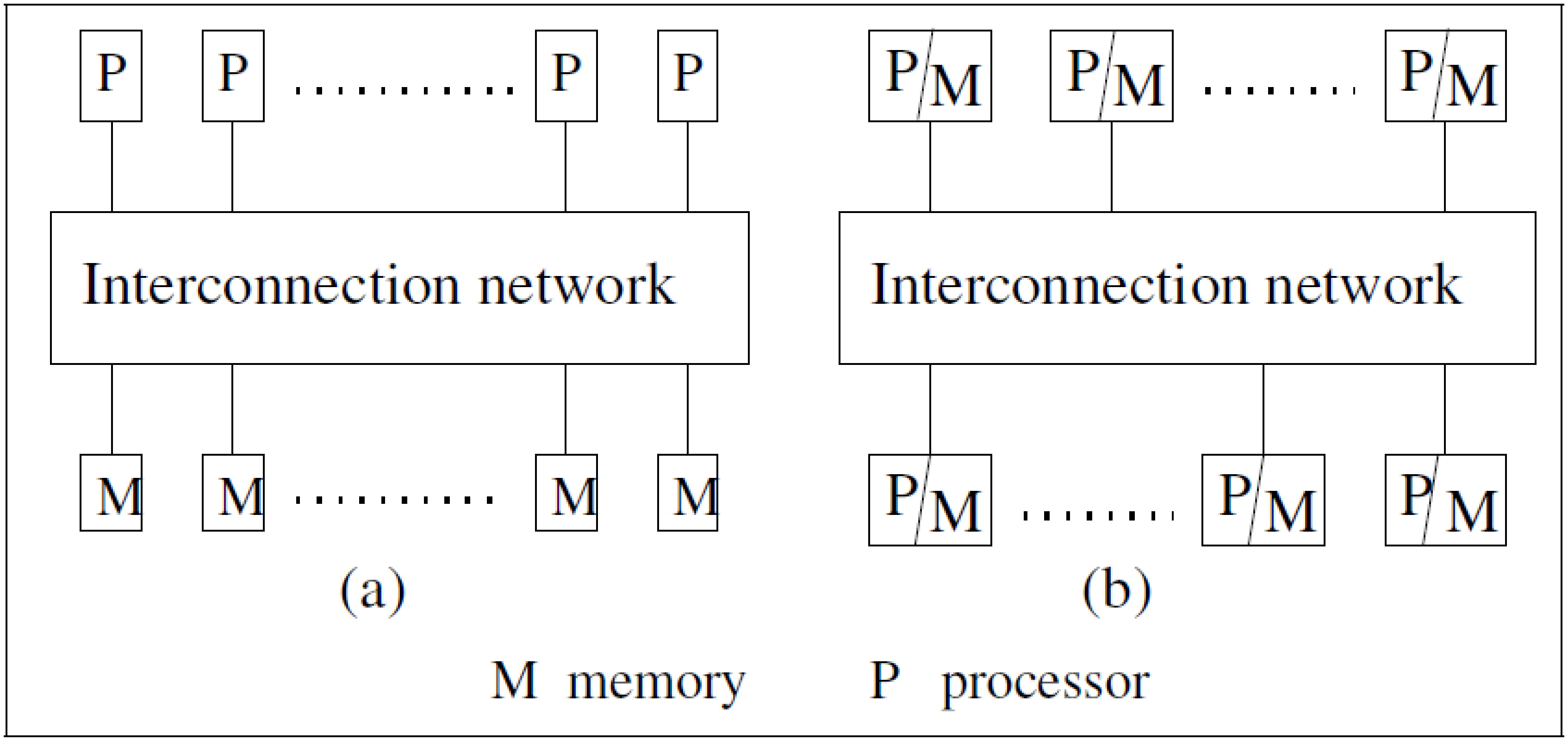
Second Semester 2014-2015

SS ZG 526: Distributed Computing

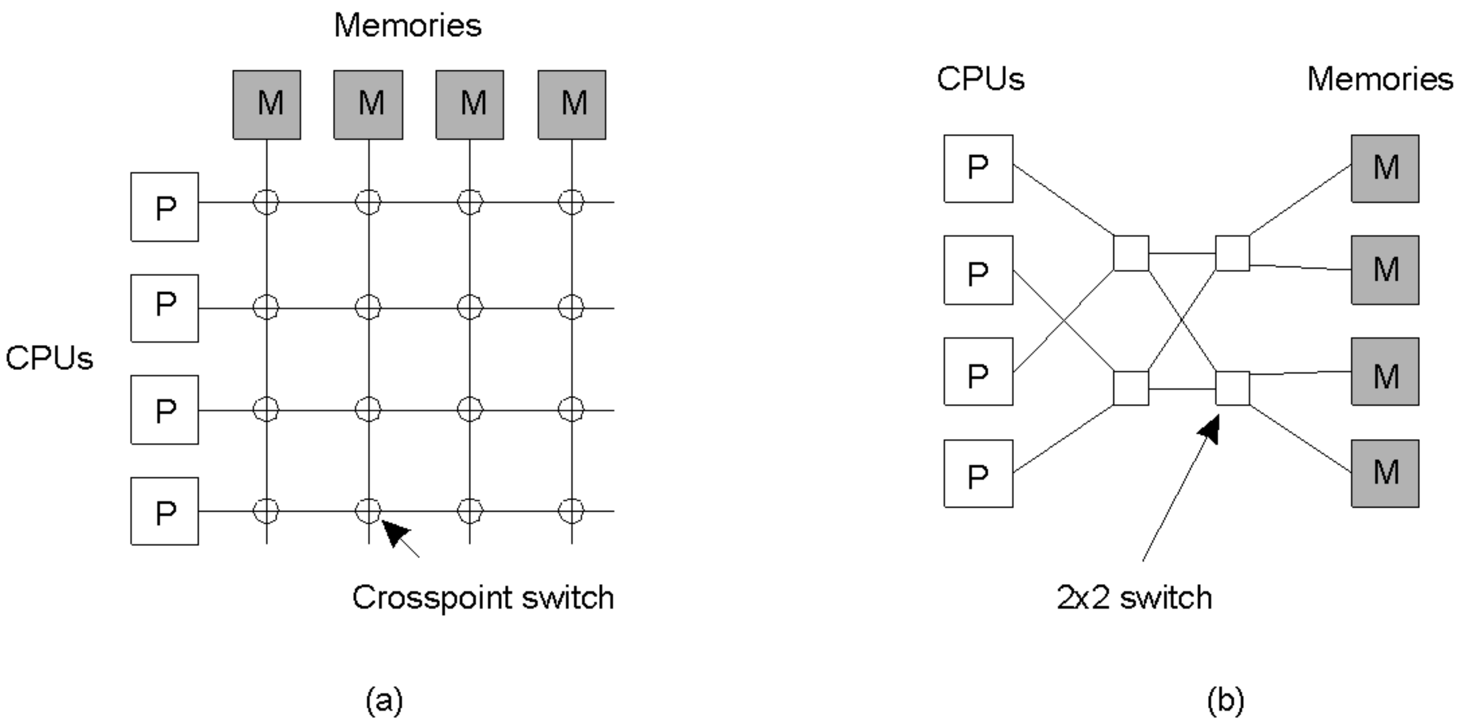
Lect 2: Communication models, and Design Issues

Chittaranjan Hota, PhD
Dept. of Computer Sc. & Information Systems
hota@hyderabad.bits-pilani.ac.in

Multiprocessor Vs Multi-computer systems



Multiprocessors



a) A crossbar switch
b) An omega switching network

Distributed Computing models

- Minicomputer model (e.g. VAXs)
  - Each computer supports many users
  - Local processing but can fetch remote data (files, databases)
- Workstation model (e.g. Athena, and Andrew)
  - Most of the work is locally done
  - Using a dfs, a user can access remote data
- Processor pool model (e.g. Amoeba is a combination of workstation and processor pool model)
  - Terminals are Xterms or diskless terminals
  - Pool of backend processors handle processing

Distributed Computing Systems: Pros and Cons

Advantages

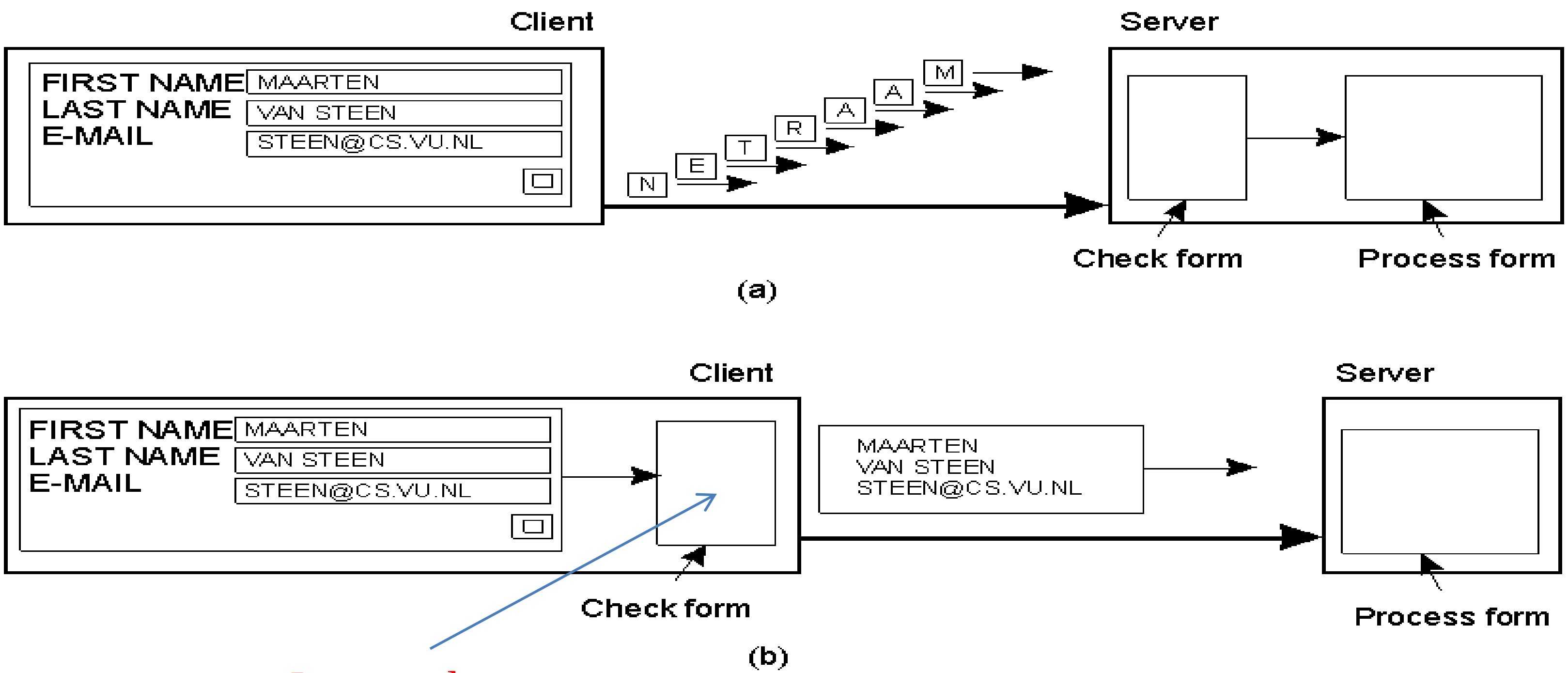
- Communication and resource sharing possible
- Economy : Price-performance
- Reliability & scalability
- Potential for incremental growth

Disadvantages

- Distribution-aware OSs and applications
- High network connectivity is essential
- Security and privacy

Design Issues: Scaling

- If possible, do asynchronous communication
  - Not always possible if the client has nothing to do
- Alternatively, by moving part of the computation



Java applets

Technique (1): Hiding latency

Scaling Technique (2): Distribution aware

Examples: DNS resolutions

More Design Issues

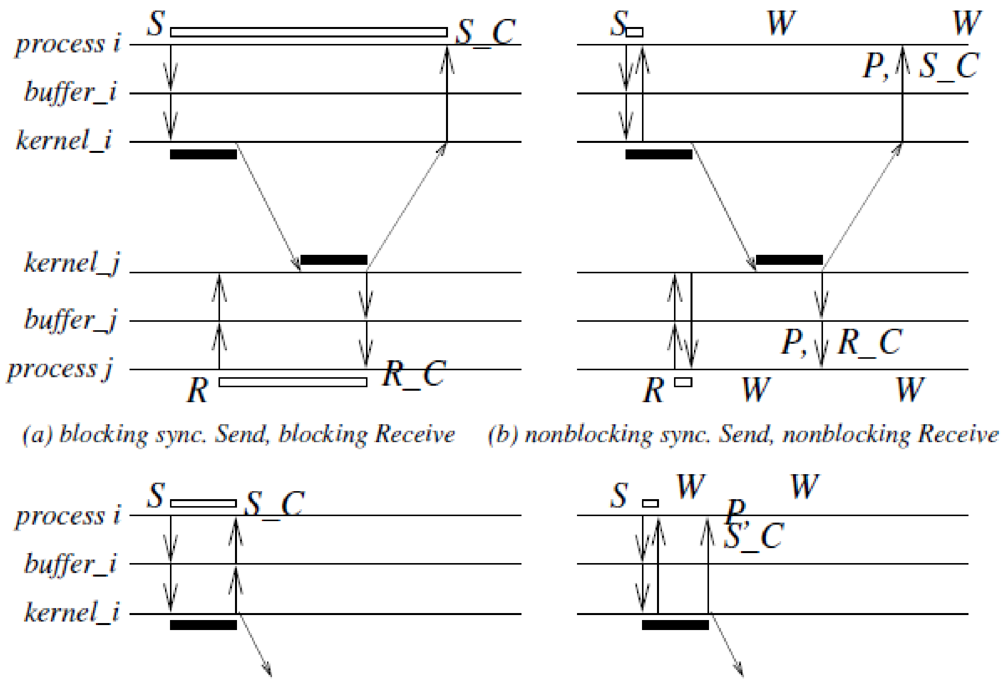
- Lack of Global Knowledge
- Naming
- Compatibility
- Process Synchronization
- Resource Management
- Security

Img. Source: www.bbc.co.uk

Communication primitives

- Synchronous (send/receive)
  - Handshake between sender and receiver
  - Send completes when Receive completes
  - Receive completes when data copied into buer
- Asynchronous (send)
  - Control returns to process when data copied out of user-specified buffer
- Blocking (send/receive)
  - Control returns to invoking process after processing of primitive (whether sync / async) completes
- Non-blocking (send/receive)
  - Control returns to process immediately after invocation
  - Send: even before data copied out of user buffer
  - Receive: even before data may have arrived from sender

Distributed Communication : Blocking /non-blocking; Synchronous/Asynchronous; send/receive primitives



(a) blocking sync. Send, blocking Receive (b) nonblocking sync. Send, nonblocking Receive

(c) blocking async. Send (d) nonblocking async. Send

▬ duration to copy data from or to user buffer

▭ duration in which the process issuing send or receive primitive is blocked

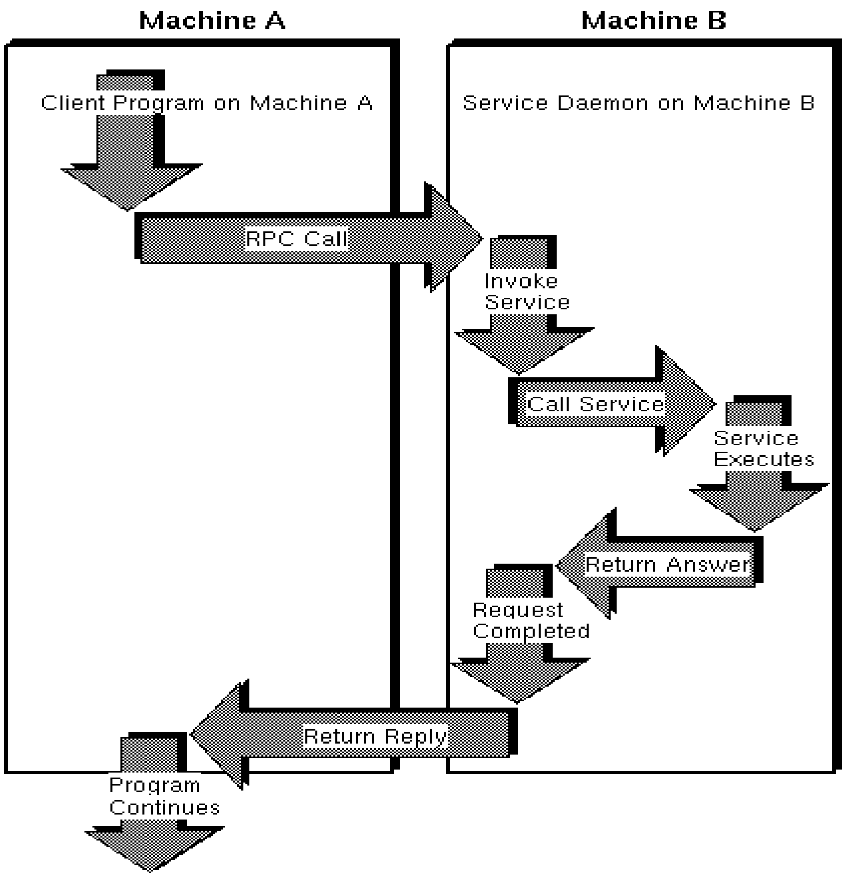
S	*Send* primitive issued	*S_C*	processing for *Send* completes
R	*Receive* primitive issued	*R_C*	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation		
W	Process may issue *Wait* to check completion of nonblocking operation		

RPC: Remote Procedure Call

- Issues:
  - identifying and accessing the remote procedure
  - parameters
  - return value
- Sun RPC
- Microsoft's DCOM
- OMG's CORBA
- Java RMI
- XML/RPC
- SOAP/.NET
- AJAX (Asynchronous Javascript and XML)

Many types

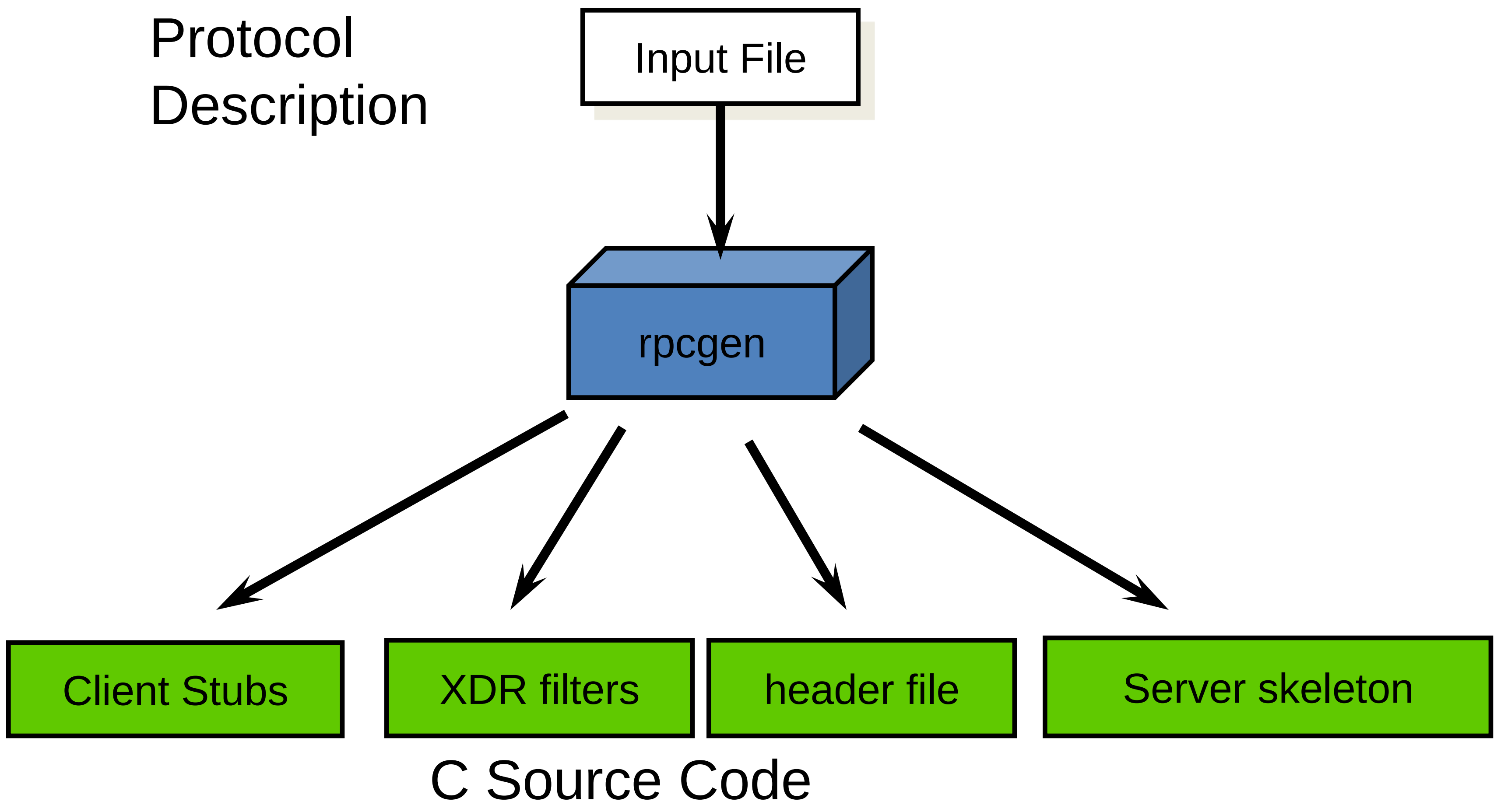
Remote Procedure Calls (RPC)



SUN RPC

```
struct square_in {
        long arg1;
};
struct square_out {
        long res1;
};
program   SQUARE_PROG {
        version SQUARE_VERS {
                square_out SQUAREPROC(square_in) = 1;
                } = 1;
} = 0x13451111;
```

Rpcgen



Protocol
Description
Input File
rpcgen
Client Stubs
XDR filters
header file
Server skeleton
C Source Code

Rpcgen continued…

bash$ rpcgen square.x

produces:

- square.h header
- square_svc.c server stub
- square_clnt.c client stub
- square_xdr.c XDR conversion routines

Function names derived from IDL function names and version numbers

Square Client: Client.c

```
#include "square.h"
#include <stdio.h>
int main (int argc, char **argv)
{ CLIENT *cl;
   square_in in;
   square_out *out;
   if (argc != 3) {  printf("client   <localhost>  <integer>"); exit (1); }
   cl = clnt_create (argv[1], SQUARE_PROG, SQUARE_VERS, "tcp");
   in.arg1 = atol (argv [2]);
   if ((out = squareproc_1(&in, cl)) == NULL)
          { printf ("Error");   exit(1); }
   printf ("Result %ld\n", out -> res1);
   exit(0);
}
```

Square server: server.c

```
#include "square.h"
#include <stdio.h>
square_out  *squareproc_1_svc (square_in *inp, struct
svc_req  *rqstp)
{
   static square_out  *outp;
   outp.res1 = inp -> arg1 * inp -> arg1;
   return (&outp);
}
```

Exe creation

- gcc –o client client.c square_clnt.c square_xdr.c –lnsl

- gcc –o server server.c square_svc.c square_xdr.c –lrpcsvc -lnsl