Network Programming

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani

Pilani Campus

Outline

Outline

- RPC
  - SUN RPC Programming
- Final Review

innovate achieve lead

BITS Pilani

Pilani Campus

RPC Programming

RPC Programming

- RPC library is a collection of tools for automating the creation of clients and servers
  - clients are processes that call remote procedures
  - servers are processes that include procedure(s) that can be called by clients
- 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 RPC library calls support UDP only
    - must use lower-level RPC library functions to use TCP
  - High-Level library calls do not support any kind of authentication

Low-level RPC Library

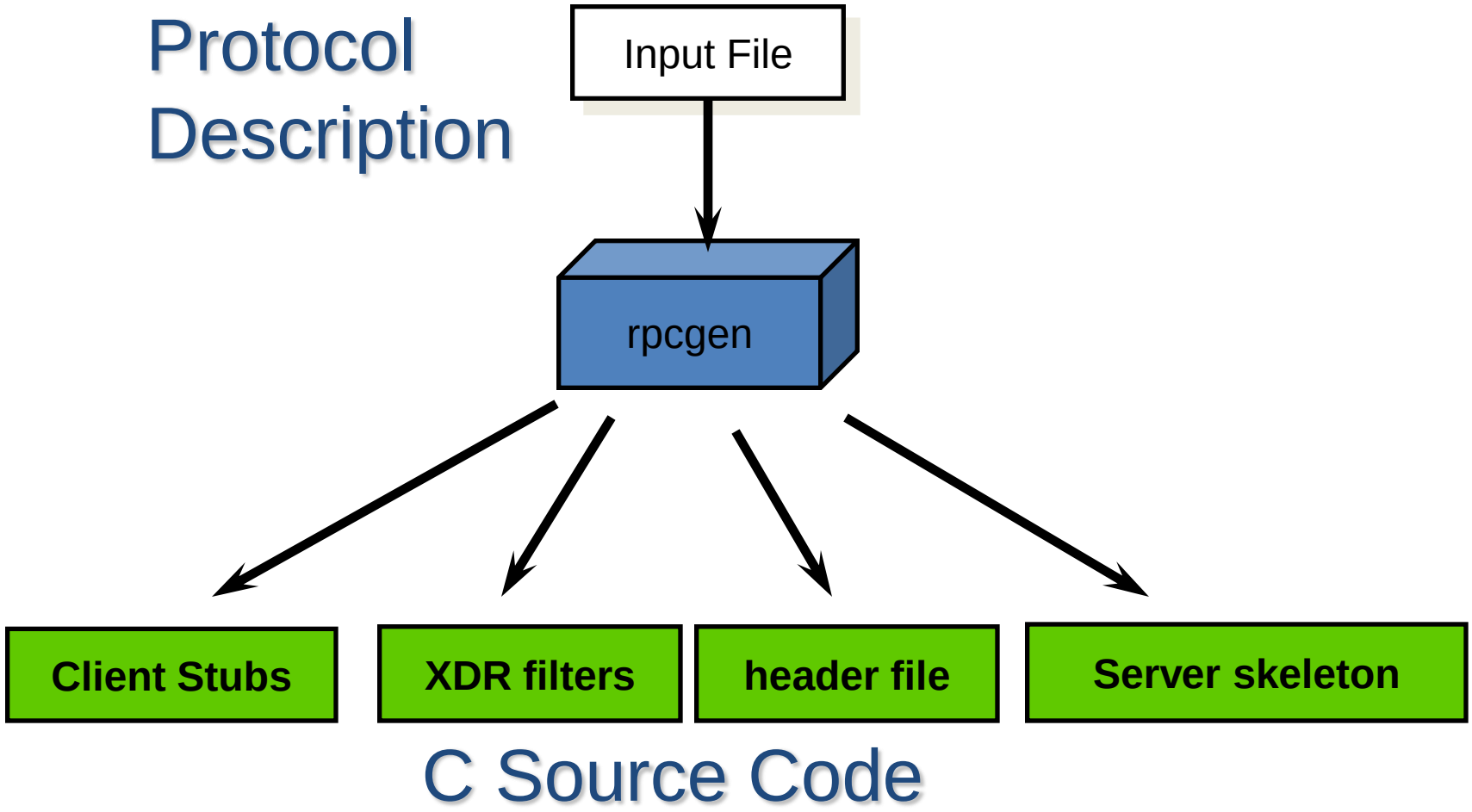
innovate achieve lead

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

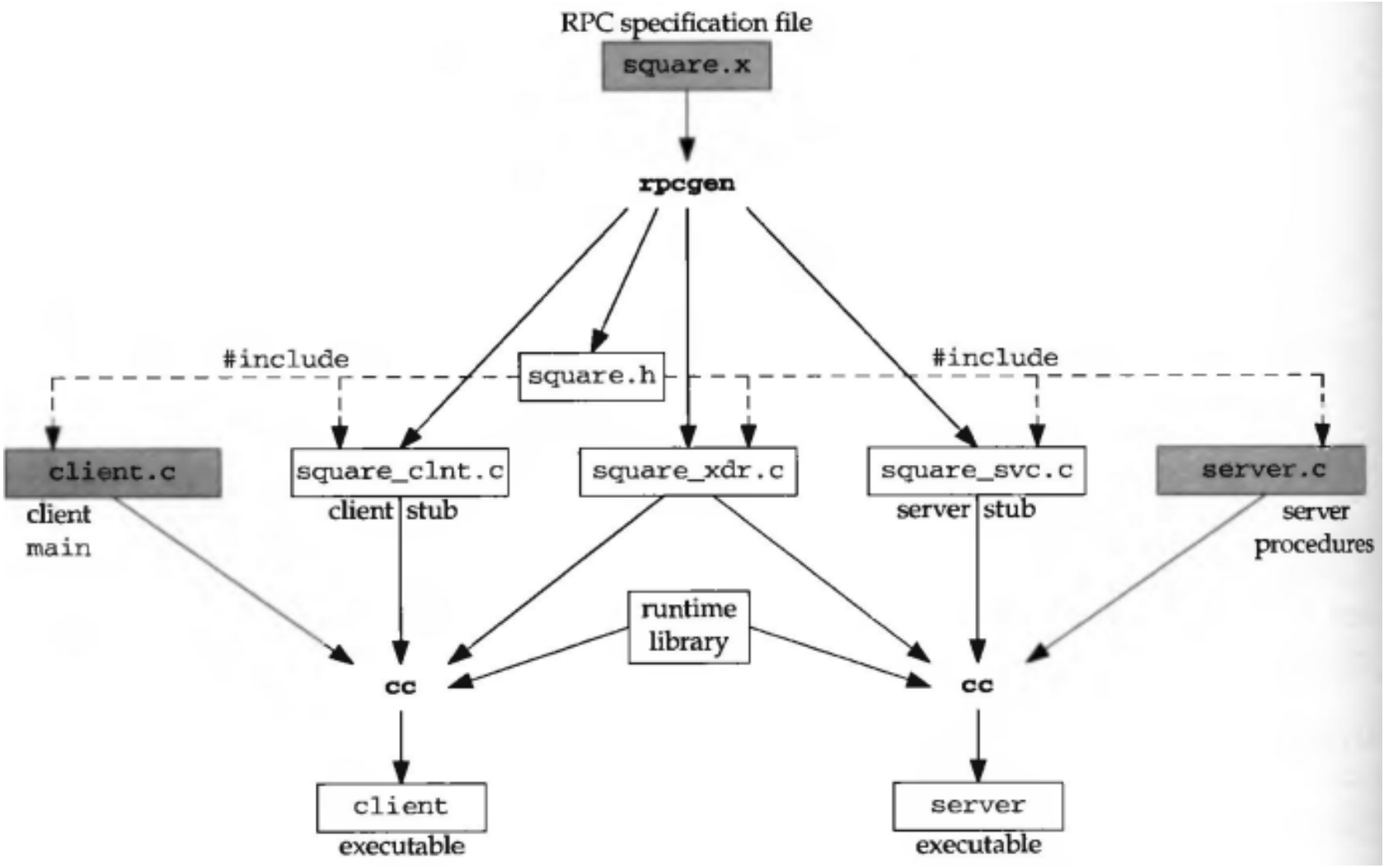
RPCGEN

- 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


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

RPCGEN


RPC specification file
square.x
rpcgen
#include
square.h
#include
client.c
client
main
square_clnt.c
client stub
square_xdr.c
square_svc.c
server stub
server.c
server
procedures
runtime
library
cc
cc
client
executable
server
executable

rpcgen Output Files

innovate achieve lead

> 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

innovate achieve lead

> 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.

RPC Programming with rpcgen

Issues:

- Protocol Definition File
  - Define argument and return value
  - define program, version, and procedure
- Client Programming
  - Creating an "RPC Handle" to a server
  - Calling client stubs
- Server Programming
  - Writing Remote Procedures

Protocol Definition File

- Description of the *interface* of the remote procedures.
  - Almost function prototypes
- Definition of any data structures used in the calls (argument types & return types)
- Can also include shared C code (shared by client and server).

Example

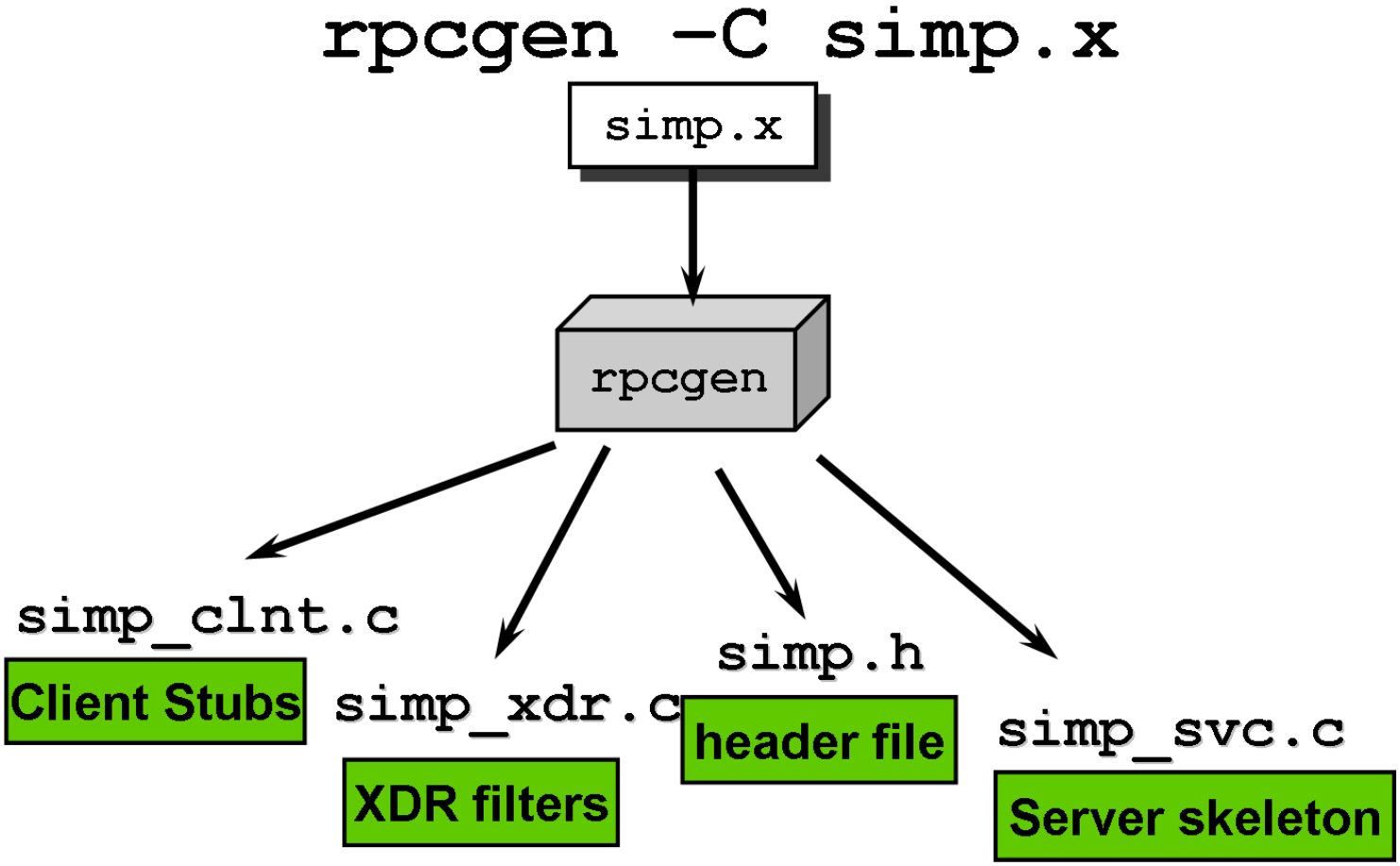
- Standalone program simp.c
  - Takes 2 integers from command line and
  - prints out the sum and difference.
- Functions:
  - int add(int x, int y);
  - int subtract(int x, int y);
- Distributed Version
  - Move the functions add() and subtract() to the server.
  - Change simp.c to be an RPC client
    - Calls stubs add_1() , subtract_1()
  - Create server that serves up 2 remote procedures
    - add_1_svc() and subtract_1_svc()

Example simp.x file

innovate achieve lead

```
1  /*To put something explicitly in the .h file produced by rpcgen
2    start it with %:
3  */
4  %#define foo 127
5  /* rpcgen just strips the '%' and puts the rest in the .h file */
6  /* here is the definition of the data type that will be passed to
7     both of remote procedures */
8  struct operands {
9      int x;
10     int y;
11 };
12 /* note that this data type will be defined in the .h file produced by
13    rpcgen. AND it will be typedef'd as well, so it is referred as
14    type 'operands', no need to use 'struct operands'.
15 */
16 /*the program, version and procedure
17 definitions. Remember this is not C, although it looks similar */
18 program SIMP_PROG {
19    version SIMP_VERSION {
20       int ADD(operands) = 1;
21       int SUB(operands) = 2;
22    } = 1;
23 } = 555555555;
```

Rpcgen Tool


rpcgen -C simp.x
simp.x
rpcgen
simp_clnt.c
Client Stubs
simp_xdr.c
XDR filters
simp.h
header file
simp_svc.c
Server skeleton

simp.h


```
1  /*Please do not edit this file. It was generated using rpcgen. */
2  #ifndef _SIMP_H_RPCGEN
3  #define _SIMP_H_RPCGEN
4  #include <rpc/rpc.h>
5  #define foo 127
6  struct operands {
7      int x;
8      int y;
9  };
10 typedef struct operands operands;
11 #define SIMP_PROG 555555555
12 #define SIMP_VERSION 1
13 #if defined(__STDC__) || defined(__cplusplus)
14 #define ADD 1
15 extern  int * add_1(operands *, CLIENT *);
16 extern  int * add_1_svc(operands *, struct svc_req *);
17 #define SUB 2
18 extern  int * sub_1(operands *, CLIENT *);
19 extern  int * sub_1_svc(operands *, struct svc_req *);
20 extern int simp_prog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);
```

- Structures are automatically typedefed.
- Creates C-type client stubs and server stubs.

Client

- This was the main program – is now the client.
- Reads 2 ints from the command line.
- Creates a RPC handle.
- Calls the remote add and subtract procedures.
- Prints the results.

Client Main Function

innovate achieve lead

```c
/* RPC client for simple addition example */
#include <stdio.h>
#include "simp.h"  /* Created for us by rpcgen - has everything we need ! */
int main( int argc, char *argv[]) {
  CLIENT *clnt;  int x,y;
  if (argc!=4) {
    fprintf(stderr,"Usage: %s hostname num1 num\n",argv[0]);
    exit(0);
  }
  /* Create a CLIENT data structure  */
  clnt = clnt_create(argv[1], SIMP_PROG, SIMP_VERSION, "udp");
  /* Make sure the create worked */
  if (clnt == (CLIENT *) NULL) {
    clnt_pcreateerror(argv[1]);    exit(1);
  }
  /* get the 2 numbers that should be added */
  x = atoi(argv[2]);
  y = atoi(argv[3]);
  printf("%d + %d = %d\n",x,y, add(clnt,x,y));
  printf("%d - %d = %d\n",x,y, sub(clnt,x,y));
  return(0);
}
```

Wrapper

```
23 /* Wrapper function takes care of calling the RPC procedure */
24 int add( CLIENT *clnt, int x, int y) {
25   operands ops;
26   int *result;
27   /* Gather everything into a single data structure to send to the server */
28   ops.x = x;
29   ops.y = y;
30   /* Call the client stub created by rpcgen */
31   result = add_1(&ops,clnt);
32   if (result==NULL) {
33     fprintf(stderr,"Trouble calling remote procedure\n");
34     exit(0);
35   }
36   return(*result);
37 }
```

- Prepare the operand.
- Call the stub.

Wrapper

```c
/* Wrapper function takes care of calling the RPC procedure */

int sub( CLIENT *clnt, int x, int y) {
  operands ops;
  int *result;

  /* Gather everything into a single data structure to send to the server */
  ops.x = x;
  ops.y = y;

  /* Call the client stub created by rpcgen */
  result = sub_1(&ops,clnt);
  if (result==NULL) {
    fprintf(stderr,"Trouble calling remote procedure\n");
    exit(0);
  }
  return(*result);
}
```

Stub for sub() function:

```c
int *sub_1(operands *argp, CLIENT *clnt)
{
    static int clnt_res;
    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call (clnt, SUB,
        (xdrproc_t) xdr_operands, (caddr_t) argp,
        (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
        TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
    return (&clnt_res);
}
```

RPC Server

- The server main is in simp_svc.c.
- simpservice.c is what we write – it holds the add and subtract procedures that simp_svc will call when it gets RPC requests.

```c
/* Definition of the remote add and subtract procedure used by
   simple RPC example
   rpcgen will create a template for you that contains much of the code
   needed in this file is you give it the "-Ss" command line arg.*/
#include <stdio.h>
#include "simp.h"
/* Here is the actual remote procedure */
/* The return value of this procedure must be a pointer to int! */
/* we declare the variable result as static so we can return a
   pointer to it */
int *
add_1_svc(operands *argp, struct svc_req *rqstp)
{   static int  result;
    printf("Got request: adding %d, %d\n",
           argp->x, argp->y);
    result = argp->x + argp->y;
    return (&result);
}
```

RPC Server

innovate achieve lead

```
22  int *sub_1_svc(operands *argp, struct svc_req *rqstp)
23  {
24      static int  result;
25      printf("Got request: subtracting %d, %d\n",
26              argp->x, argp->y);
27      result = argp->x - argp->y;
28      return (&result);
29  }
```

```
struct svc_req {
   u_long               rq_prog;     /* program number */
   u_long               rq_vers;     /* version number */
   u_long               rq_proc;     /* procedure number */
   struct opaque_auth   rq_cred;     /* raw credentials */
   caddr_t              rq_clntcred; /* cooked credentials (read-only) */
   SVCXPRT             *rq_xprt;     /* transport handle */
};
```

RPC Authentication

innovate achieve lead

stem	Authentication technique	Comments
AUTH_NONE	None	No authentication. Anonymous access.
AUTH_UNIX	RPC client sends the Unix UID and GIDs for the user	Not secure. Server implicitly trusts that the user is who the user claims to be.
AUTH_DES	Authentication based on public key cryptography and DES	Reasonably secure.
AUTH_KERB	Authentication based on Kerberos	Reasonably secure, but requires that you properly set up a Kerberos server. AUTH_KERB is not universally available.

XDR Data Types

	RPC specification file (.x)	C header file (.h)
1	`const` *name* `=` *value*`;`	`#define` *name value*
2	`typedef` *declaration*`;`	`typedef` *declaration*`;`

3	`char` *var*`;`
	`short` *var*`;`
	`int` *var*`;`
	`long` *var*`;`
	`hyper` *var*`;`
		`char` *var*`;`
		`short` *var*`;`
		`int` *var*`;`
		`long` *var*`;`
		`longlong_t` *var*`;`

4	`unsigned char` *var*`;`
	`unsigned short` *var*`;`
	`unsigned int` *var*`;`
	`unsigned long` *var*`;`
	`unsigned hyper` *var*`;`
		`u_char` *var*`;`
		`u_short` *var*`;`
		`u_int` *var*`;`
		`u_long` *var*`;`
		`u_longlong_t` *var*`;`

5	`float` *var*`;`
	`double` *var*`;`
	`quadruple` *var*`;`
		`float` *var*`;`
		`double` *var*`;`
		`quadruple` *var*`;`

6	`bool` *var*`;`	`bool_t` *var*`;`
7	`enum` *var* `{` *name* `=` *const*`, ... };`	`enum` *var* `{` *name* `=` *const*`, ... };` `typedef enum` *var var*`;`

XDR Data Types

10	`string var<m>;`	`char *var;`
11	`datatype var[n];`	`datatype var[n];`

12	`datatype var<m>;`	`struct {`
		`  u_int var_len;`
		`  datatype *var_val;`
		`} var;`

13	`struct var { members ... };`	`struct var { members ... };` `typedef struct var var;`

14	`union var switch (int disc) {`
	`  case discvalueA: armdeclA;`
	`  case discvalueB: armdeclB;`
	`  ...`
	`  default: defaultdecl;`
	`};`
		`struct var {`
		`  int disc;`
		`  union {`
		`  armdeclA;`
		`  armdeclB;`
		`  ...`
		`  defaultdecl;`
		`  } var_u;`
		`};`
		`typedef struct var var;`

15	`datatype *name;`	`datatype *name;`

Other RPC Systems

- Sun's RPC is not unique. A different RPC system is used by the Open Software Foundation's Distributed Computing Environment (DCE).
- Another RPC system was proposed by the Object Management Group. Named CORBA (Common Object Request Broker Architecture), this system is optimized for RPC between object-oriented programs written in C++ or SmallTalk.
- Java programmers use Remote Method Invocation (RMI).

innovate achieve lead

BITS Pilani

Pilani Campus

Final Review

Topics

- UDP client-server
  - Connect
  - Concurrency
- Domain name conversion
  - getaddrinfo()
- I/O Models
  - Non-blocking I/O
  - I/O Multiplexing
  - Signal driven I/O
  - Asynchronous I/O
- Unix Domain Protocol
- Pthreads
- Client Design
- Server Design
  - Preforking
  - Prethreading
  - Single process Models
- Raw Sockets
- Datalink access
- Broadcasting Multicasting
- RPC

Previous Question paper

innovate achieve lead

- Explain how would you solve the following problems using the concepts learnt during this course. Give a piece of code wherever applicable. Single line answers are not acceptable.
- In a LAN, you want to discover whether there is any machine running a service at port number 111. How will you do it?
  - Broadcasting service must be used.
  - Write the steps involved in broadcasting.
- You want to count the number of packets received and sent by your system on a given network interface. How will you do it?
  - Use libpcap/BPF/DLPI. open the device. read the packets and cout.
- You want to put a timeout of 10 nano seconds on a UDP recvfrom call. How will you do it?
  - Use pselect.

Q1

- You are given an already open descriptor (say through a Unix domain socket) and asked to identify whether it is a socket descriptor or a file descriptor. How will you do it?
  - a call like lseek will fail on socket descriptor.
- Suppose you want to do SYN flooding attack on a TCP server. Without modifying the TCP/IP module on your system, how will you do it?
  - Use raw sockets to write a TCP SYN segment.

Q2

- Write a program that takes a host name such as "www.bits-pilani.ac.in" as command line argument and prints all the aliases that exist for this name.

```
1  int main(int argc, char **argv)
2  {
3      char            *ptr, **pptr;
4      char            str[INET_ADDRSTRLEN];
5      struct hostent  *hptr;
6      while (--argc > 0) {
7          ptr = *++argv;
8          if ( (hptr = gethostbyname(ptr)) == NULL) {      \\2M
9              err_msg("gethostbyname error for host: %s: %s",
10                     ptr, hstrerror(h_errno));
11             continue;
12         }
13
14         for (pptr = hptr->h_aliases; *pptr != NULL; pptr++) \\2M
15             printf("\talias: %s\n", *pptr);
16
17     }
18     exit(0);
19 }
```

Q3

- User wants to join 3 multicast session with addresses 239.255.1.1 at port 5000, 239.255.2.1 at port 6000 and 239.255.3.1 at port 7000. Write a program that joins these groups and receives the messages from these groups as and when they arrive and display to the console.

```
1  main(int argc)
2  {  struct sockaddr_in addr;
3     int addrlen, sock, cnt;
4     struct ip_mreq mreq;
5     char message[50];
6     /* set up 3 sockets */
7     sock1 = socket(AF_INET, SOCK_DGRAM, 0);
8     sock2 = socket(AF_INET, SOCK_DGRAM, 0);
9     sock3 = socket(AF_INET, SOCK_DGRAM, 0);
10     bzero((char *)&addr, sizeof(addr));
11    addr.sin_family = AF_INET;
12    addr.sin_addr.s_addr = htonl(INADDR_ANY);
13    addr.sin_port = htons(5000);
14    addrlen = sizeof(addr);
15 if (bind(sock1, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
16          perror("bind");     exit(1);      }
17    addr.sin_port = htons(6000);
18 if (bind(sock2, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
19          perror("bind");     exit(1);     }
20    addr.sin_port = htons(7000);
21 if (bind(sock3, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
22          perror("bind");     exit(1);      }
```

Q3

```
25      mreq.imr_multiaddr.s_addr = inet_addr(239.255.1.1);
26        mreq.imr_interface.s_addr = htonl(INADDR_ANY);
27        if (setsockopt(sock1, IPPROTO_IP, IP_ADD_MEMBERSHIP,
28               &mreq, sizeof(mreq)) < 0) {
29       perror("setsockopt mreq");
30       exit(1);
31        }
32      mreq.imr_multiaddr.s_addr = inet_addr(239.255.2.1);
33        mreq.imr_interface.s_addr = htonl(INADDR_ANY);
34        if (setsockopt(sock2, IPPROTO_IP, IP_ADD_MEMBERSHIP,
35               &mreq, sizeof(mreq)) < 0) {
36       perror("setsockopt mreq");
37       exit(1);
38        }
39      mreq.imr_multiaddr.s_addr = inet_addr(239.255.3.1);
40        mreq.imr_interface.s_addr = htonl(INADDR_ANY);
41        if (setsockopt(sock3, IPPROTO_IP, IP_ADD_MEMBERSHIP,
42               &mreq, sizeof(mreq)) < 0) {
43       perror("setsockopt mreq");
44       exit(1);      }
```

Q3

innovate achieve lead

```
47  for(;;){
48      fd_set      rset;
49      char        sendline[MAXLINE], recvline[MAXLINE];
50
51      FD_ZERO(&rset);
52      for ( ; ; ) {
53          FD_SET(sock1, &rset);
54          FD_SET(sock2, &rset);
55          FD_SET(sock3, &rset);
56
57          maxfdp1 = sock3 + 1;
58          Select(maxfdp1, &rset, NULL, NULL, NULL);
59
60          if (FD_ISSET(sock1, &rset)) {   /* socket1 is readable */
61              Fputs(recvline, stdout);
62          }
63          if (FD_ISSET(sock2, &rset)) {   /* socket2 is readable */
64              Fputs(recvline, stdout);
65          }
66          if (FD_ISSET(sock3, &rset)) {   /* socket3 is readable */
67              Fputs(recvline, stdout);
68          }
```

Q4

innovate achieve lead

- Write a RPC client and server using high-level API of SUN RPC framework. Server maintains a database of railways. List of trains, fare etc. When user types a train id, client fetches the details from the server and displays them on the screen. User should be able to perform transactions
  - bookTicket(). booking of a ticket by giving train no, date, from station, to station, and passenger name, and age, and sleeper or 3rd AC or 2nd AC. It should return the PNR number.
  - cancelTicket(). It should take PNR number and cancel the ticket.
  - displayInfo(): It should take the train no, start station, and end station and date. It returns vacancies in sleeper and 3rd AC.

Develop protocol file that is input for the *rpcgen* tool.

Q4

innovate achieve lead

```
1  struct boperands {
2      int train no;
3      string date<20>;
4      string fs<50>;
5      string ts<50>;
6      string pname<50>;
7      int age;
8      string type<10>;
9  };
10 struct coperands {
11     long pnr;
12 };
13 struct doperands {
14     int train no;
15     string date<20>;
16     string fs<50>;
17     string ts<50>;
18 };
```

```
20 struct rop{
21     int sleeper_vac;
22     int 3ac_vac;
23 };
24
25 program RAIL_PROG {
26    version RAIL_VERSION {
27         long bookTicket(boperands) = 1;
28         int cancelTicket(coperands) = 2;
29         rop displayInfo(doperands) = 3;
30    } = 1;
31 } = 555575555;
```

Q5

- Write a mini inetd daemon server. Assume that the following array of structures is available in your program with data for each service. [10]
- struct service{
  - int16_t port;
  - char protocol[MAX]; //tcp or udp
  - char concurrent[MAX]; //yes or no
  - char serverprogram_path[MAX]; //path of the executable
- };

Q5

innovate achieve lead

```
12  main(){
13      struct service serv[10];
14      fp=fopen("conf.txt","r");
15  \\reading from file and creating sockets 4M
16      while(fgets(line,100,fp)!=NULL){
17      sscanf(line,"%s %d %s %s %s",serv[number].name,&serv[number].port,
18      serv[number].protocol,serv[number].concurrent,serv[number].serverprogram_path);
19
20      if(strcmp(serv[number].protocol,"udp")==0)
21          serv[number].socket = socket(AF_INET, SOCK_DGRAM, 0);
22      else if(strcmp(serv[number].protocol,"tcp")==0)
23          serv[number].socket = socket(AF_INET, SOCK_STREAM, 0);
24      else printf("protocol doesn't exist!!!");
25     if(maxfd<serv[number].socket) maxfd=serv[number].socket;
```

```
27      bzero((char *)&serv[number].addr, sizeof(serv[number].addr));
28      serv[number].addr.sin_family = AF_INET;
29      serv[number].addr.sin_addr.s_addr = htonl(INADDR_ANY);
30      serv[number].addr.sin_port = htons(serv[number].port);
31     serv[number].len = sizeof(serv[number].addr);
32     if (bind(serv[number].socket, (struct sockaddr *) &serv[number].addr,
33     sizeof(serv[number].addr)) < 0) {
34          perror("bind");exit(1);      }
35    if(strcmp(serv[number].protocol,"tcp")==0){
36              if (listen(serv[number].socket, 10) < 0)
37                  perror("listen() failed");
38          }
39  number++;
40  }
```

Q5

innovate achieve lead

```
42  FD_ZERO (&allset);
43  for (i = 0; i < number; i++){
44      FD_SET(serv[i].socket, &allset);
45  }
46  for(;;){
47       rset = allset;
48       nready = select (maxfd + 1, &rset, NULL, NULL, NULL); \\1M
49           for(i=0;i<number;i++){
50           if (FD_ISSET (serv[i].socket, &rset)){
51           if(strcmp(serv[i].protocol,"tcp")==0){
52           len = sizeof(cli);
53          \\1M
54              if ((connfd = accept(serv[i].socket, (struct sockaddr *) &cli,
55                                                  &len)) < 0){
56              //printf("failure!!");
57              perror("accept() failed");
58          }
```

Q5

innovate achieve lead

```
61          pid=fork();
62                  if(pid==0){
63                      int sockc;
64                      sockc=strcmp(serv[i].protocol,"tcp")==0?connfd:serv[i].socket;
65                      for(j=0;j<64;j++){
66                         if(j==sockc) continue;
67                         close(j);
68                      }
69                      dup2(sockc,0);
70                      dup2(sockc,1);
71                      dup2(sockc,2);
72                      close(sockc);
73                      setsid();
74                      if(execl(serv[i].serverprogram_path,serv[i].name,(char *)NULL)==-1)
75                          perror("exec");
76                  }
77                  else{
78                      if(strcmp(serv[i].protocol,"tcp")==0){
79                          close(connfd);
80                          if(strcmp(serv[i].concurrent,"no")==0){
81                              waitpid(pid,0,0);
82                          }
```

Acknowledgements

Q&A

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You