Operating Systems

RPC: Processes

Chapter 3: Processes: Outline

- Process Concept: views of a process
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Inter Process Communication (IPC)
  - Local
    - Pipe
    - Shared Memory
    - Messages (Queues)
  - Remote
    - Lower Level: Sockets, MPI, Myrinet
    - Higher Level: RPC, RMI, WebServices, CORBA,

Client-Server Remote Machine Communication Mechanisms

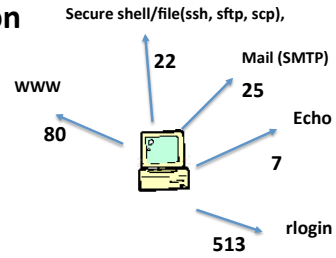
- Socket communication (Possible bonus project)
- Remote Procedure Calls (Project due next week).
- Remote Method Invocation (Briefly, on your own)

Remote Procedure Calls (RPC)

- Inter-machine process to process communication
  - (abstract) **procedure calls** across a network:
  - FunctionCall [address] [parameters]
  - Address – machine [& port]
  - rusers, rstat, rlogin, rup => daemons at ports
    - Registered library calls (port mapper)
    - Many are now disabled due to security concerns (here)
  - Hides message passing I/O from programmer
- Looks (almost) like a procedure call -- but client invokes a procedure on a server.
  - Pass arguments – get results
  - Fits into high-level programming language constructs
  - Well understood

Address: IP_number[:Port_number]

- Identifies the **ultimate destination**
- IP addresses identify **hosts**
  - **127.0.0.1, 172.20.10.15, 128.192.101.135**
  - **{ingrid:509} nslookup nike.cs.uga.edu**
  - **ifconfig**
- Host has many applications → ports
- Ports (16-bit identifier) 1-65,535 (about 2000 are reserved).



Well-known	1-1,023
Registered	1,024-49,151
Dynamic	49,152-65,535

https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

rlogin [nike.cs.uga.edu:513]

- Problems: Passwords transmitted unencrypted.
- .rlogin/.rhosts files
  - Allow logins without a password

```
{nike:maria:4} tail ~/.rhosts
herc                    ingrid
odin                    maria
nike                    maria
nike.cs.uga.edu         maria
vcf5                    maria
vcf4                    maria
vcf3                    maria
vcf2                    maria
vcf1                    maria
vcf0                    maria
{nike:maria:5}
```


RPC Calls : Portmapper

- RPC applications picks any available port then registers with a portmapper daemon

Remote Procedure Calls (RPC)

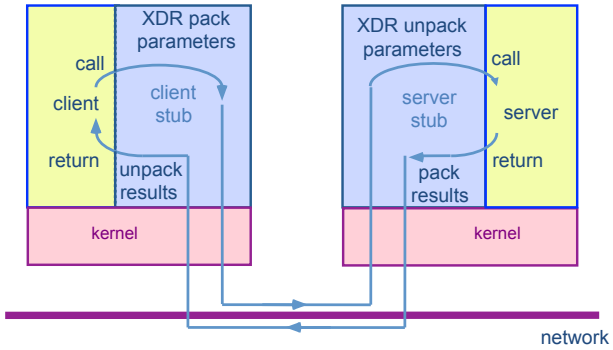
- RPC High level view:
  - Calling process attempt to call a 'remote' routine on server
  - Calling process (client) is suspended
  - Parameters are passed across network to a process server
  - Server executes procedure
  - Return results across network
  - Calling process resumes

Remote Procedure Calls

- Usually built on top sockets (UDP)
- stubs – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and marshalls the parameters.
- The server-side stub receives this message, unpacks the 'marshalled' parameters, and then performs the procedure call on the server.

Association 5 tuple *{protocol, local-address, local-process, foreign-address, foreign-process}*

Client/Server Model Using RPC



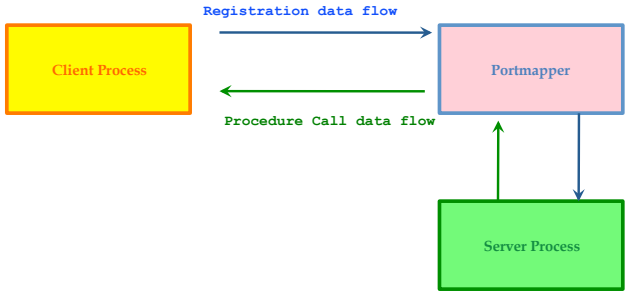
Each RPC invocation by a client process calls a ***client stub***, which builds a message and sends it to a ***server stub***

- The server stub uses the message to generate a local procedure call to the server
- If the local procedure call returns a value, the server stub builds a message and sends it to the client stub, which receives it and returns the result(s) to the client

RPC Association Between Machines

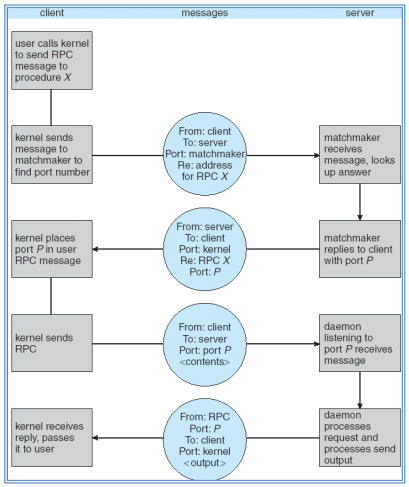
- Association between remote and local host
  - 5 tuple
    - *{protocol, local-address, local-process, foreign-address, foreign-process}*
    - *Protocol : transport protocol typically TCP or UDP, needs to be common between hosts*
    - *Local/foreign address: Typically the IP address*
    - *Local/foreign process: Typically the port number (not PID)*

Binding



- RPC application is packed into a program and is assigned an identifier (Port)
- Portmap : allocate port numbers for RPC programs

Execution of RPC



Remote Procedure Calls

- Machine independent representation of data:
  - Differ if most [or least] significant byte is in the high memory address
  - External data representation (XDR)
    - Allows more complex representation that goes beyond:
      - htonl() routines.
- Fixed or dynamic address binding
  - Dynamic: Matchmaker daemon at a fixed address (given name of RPC returns port of requested daemon)

Hide Complexity
Program to generate code

- **`rpcgen`** generates C code from a file written in 'RPC language' avoids programmer to worry about networking details
  - Stylistics – end with an X.

    `<name>.x`, e.g., **`avg.x`**
  - **`rpcgen avg.x`**

- **Leaves the programmer with 3 tasks:**
  - **`avg.x`**
  - Create Client routine (main program on local host), then run it.
    - **`ravg <host> <parameters>`**
    - **`ravg localhost 1 2 3 4`**
    - **`ravg vcf4 1 2 3 4`**
    - **`ravg vcf4 $RANDOM $RANDOM`**
  - Create Server program (e.g., actual code to compute something, e.g., an average), then run it:
    - **`avg_proc &`**
    - **`rpcinfo –p localhost`**

https://docs.oracle.com/cd/E19683-01/816-1435/rpcgenpguide-21470/index.html

http://www.linuxjournal.com/article/2204?page=0,1

Tutorial (linux journal)

- `rpcgen` generates C code from a file written in 'RPC language' `<name>. x`, e.g., `avg.x`

Default output rpcgen	Syntax	Example
Header file	`<name>.h`	`avg.h`
XDR data type translate routines (from type in .h file)	`<name>_xdr.c`	`avg._xdr.c`
stub program for server	`<name>_svc.c`	`avg_svc.c`
stub program for client	`<name>_clnt.c`	`avg_clnt.c`

- (Create these) Application programmer (you) write code for:
  - Client routine (main program)
    - `ravg <host> <parameters>`
  - Server program (e.g., actual code to compute average)
    - `avg_proc.c`

Application Routines of Interest

- Server Routine:
  - average_1_svc (input_data,):
    - A avg_proc.c routine that is called from the server stub that was generated by rpcgen
- Client Routine:
  - average_prog_1()
    - Local routine that parse parameter and that ultimately calls a 'local' average_1 routine from generated code in avg_clnt.c that packs parameters (also uses routines in avg_xdr.c and sends code to server.

avg.x : RPC language file

```
const MAXAVGSIZE  = 200;
struct input_data
  {
  double input_data<200>;
  };

typedef struct input_data input_data;

program AVERAGEPROG {
    version AVERAGEVERS {
        double AVERAGE(input_data) = 1;
    } = 1; /* version */
} = 22855; /* 'port number' */
```


ravg.c : Client Program(1)

```
/* client code - calls client stub, xdr client, xdr xerver, server stub, server routine */
#include "avg.h"        /* header file generated by rpcgen */
#include <stdlib.h>

/* local routine client prototype can be whatever you want */
void averageprog_1( char* host, int argc, char *argv[] )
{
   CLIENT       *clnt; /* client handle, rpc.h */
   double       f, :*result_1, *dp,
   char         *endptr;
   int i;
   input_data  average_1_arg; /* input_data rpc struct */

   average_1_arg.input_data.input_data_val = (double*) malloc(MAXAVGSIZE* sizeof(double));

   dp = average_1_arg.input_data.input_data_val;  /* ptr to beginning of data */
   average_1_arg.input_data.input_data_len = argc - 2;  /* set number of items */

   for( i = 1; i <= (argc - 2); i++ )
   {    /* str to d ASCII string to floating point nubmer */
        f = strtod( argv[i+1], &endptr);
        printf("value   = %e\n", f);
        *dp = f;
        dp++;
   }
```

ravg.c : Client Program (2)

```
  /*  clnt_create( host, program, version, protocol)
    *   generic client create routine from rpc library
    * program = AVERAGEPROG is the number 22855
    * version = AVERAGEVERS is 1
    * protocol = transfer protocol */
   clnt = clnt_create( host, AVERAGEPROG, AVERAGEVERS, "udp" );
   if (clnt == NULL)
   {  clnt_pcreateerror( host ); /* rpc error library */
      exit(1);
   }
   /* now call average routine 'just' like a local routine, but this will now go over
  network
    * average_1 is definined in the client stub in avg_clnt.c that was generated by rpcgen
    * send in ptr to the parameters or args  in first field, and client handle in second
    * field (created in clnt_create ) average_1 ultimately calls clnt_call() macro see
    * man rpc, then calls the remote routine associated with the client handle
    * so AVERAGEPROG, VERSION */
   result_1 = average_1( &average_1_arg, clnt );
   if (result_1 == NULL)
      {
      clnt_perror(clnt, "call failed:");
      }

   clnt_destroy( clnt );
   printf( "average = %e\n",*result_1 );
} /* end average_1 prodedure */ /* next slide main() */
```

ravg.c : Client Program (3)

```
int main( int argc, char* argv[] )
{
   char *host;

   /* check correct syntax */
   if( argc < 3 )
   {
        printf( "usage: %s server_host value ...\n", argv[0]);
        exit(1);
   }

   if( argc > MAXAVGSIZE + 2 )
   {
        printf("Two many input values\n");
        exit(2);
   }

   /* host name is in first parameter (after program name) */
   host = argv[1];
   averageprog_1( host, argc, argv);
}
```

avg_proc.c : Server Program (1)

```
#include <rpc/rpc.h>
#include "avg.h"  /* avg.h generated rpcgen */
#include <stdio.h>

/* run locally on 'server' called by a remote client. */
static double sum_avg;

/* routine notice the _1 the version number and  notice the client handle, not used here,
  but
 * still needs to be  a parameter */
double * average_1( input_data *input, CLIENT *client)
  {
  /* input is parameters were marshaled by generated routine */
  /*  a pointer to a double, set to beginning of data array  */
  double *dp = input->input_data.input_data_val;
  u_int i;
  sum_avg = 0;
  for( i = 1; i <= input->input_data.input_data_len; i++ ) /* iterate over input */
    {
    sum_avg = sum_avg + *dp;  /* add what ptrs points  to ( '*' gets content ) */
    dp++;
    }

  sum_avg = sum_avg / input->input_data.input_data_len;
  return( &sum_avg );
} /* end average_1 */ /* next is routine called from server stub generated by rpcgen */
```

avg_proc.c : Server Program (1)

```
#include <rpc/rpc.h>
#include "avg.h"  /* avg.h generated rpcgen */
#include <stdio.h>

/* run locally on 'server' called by a remote client. */
static double sum_avg;

/* routine notice the _1 the version number and  notice the client handle, not used here,
  but
 * still needs to be  a parameter */
double * average_1( input_data *input, CLIENT *client)
  {
  /* input is parameters were marshaled by generated routine */
  /*  a pointer to a double, set to beginning of data array  */
  double *dp = input->input_data.input_data_val;
  u_int i;
  sum_avg = 0;
  for( i = 1; i <= input->input_data.input_data_len; i++ ) /* iterate over input */
    {
    sum_avg = sum_avg + *dp;  /* add what ptrs points  to ( '*' gets content ) */
    dp++;
    }

  sum_avg = sum_avg / input->input_data.input_data_len;
  return( &sum_avg );
} /* end average_1 */ /* next is routine called from server stub generated by rpcgen */
```

avg_proc.c : Server Program (2)

```
/*
 * server stub 'average_1_svc function handle called in avg_svc that was
 * generated by rpcgen
 * FYI:
 *   result = (*local)((char *)&argument, rqstp);
 *   where local is (char *(*)(char *, struct svc_req *)) average_1_svc;
 */

double * average_1_svc(input_data *input, struct svc_req *svc)
  {
  CLIENT *client;
  return( average_1( input, client) );
  }
```

Compilation on client

```
rpcgen avg.x  # generates:
 # avg_clnt.c, avg_svc.c, avg_xdr.c, avg.h
gcc ravg.c -c        # -c generates .o files
gcc avg_clnt.c -c
gcc avg_xdr.c -c
gcc -c ravg ravg.o avg_clnt.o avg_xdr.o -lnsl
```


Compilation on server

```
rpcgen avg.x  # generates:
 # avg_clnt.c, avg_svc.c, avg_xdr.c, avg.h
gcc avg_proc.c -c
gcc avg_svc.c -c
gcc -o avg_svc avg_proc.o avg_svc.o avg_xdr.o -lnsl
```


~/.rhost

- Directly under your home directory on each machine (client and server) create a file named:

  ~/.rhost

- Add two or more lines in the format:

  <machine_name> <login name>

- For end part of my file:

```
{nike:maria:4} tail ~/.rhosts
herc                    ingrid
odin                    maria
nike                    maria
nike.cs.uga.edu         maria
vcf5                    maria
vcf4                    maria
vcf3                    maria
vcf2                    maria
vcf1                    maria
vcf0                    maria
{nike:maria:5}
```


Running:

- Start server avg_svc on node 4 on nike to sit and wait for clients to connect.
- Run client ravg on node 5 on nike and send average request:

```
/Users/ingrid — ssh maria@nike.cs.uga.edu
{vcf4:maria:43} ls
avg_clnt.c  avg_proc.o  avg.x          index.html  ravg
avg_clnt.o  avg_svc     avg_xdr.c      Makefile    ravg.c
avg.h       avg_svc.c   avg_xdr.o      m-rusers    ravg.o
avg_proc.c  avg_svc.o   DIRECTORY.html  m-rusers.c  RPCMakefile
{vcf4:maria:44} avg_svc &
[1] 27754
{vcf4:maria:45}
```

```
{vcf5:maria:76} ravg vcf4 1 2 3 4
value   = 1.000000e+00
value   = 2.000000e+00
value   = 3.000000e+00
value   = 4.000000e+00
average = 2.500000e+00
{vcf5:maria:77}
```

Resources

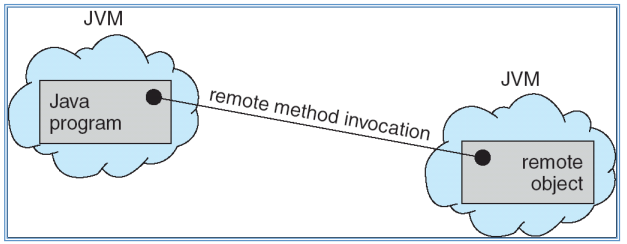
1. http://users.cs.cf.ac.uk/Dave.Marshall/C/node33.html
2. http://users.cs.cf.ac.uk/Dave.Marshall/C/node34.html
   alternate:
   https://docs.oracle.com/cd/E19683-01/816-1435/rpcgenpguide-21470/index.html
3. http://users.cs.cf.ac.uk/Dave.Marshall/C/node27.html
4. http://www.linuxjournal.com/article/2204?page=0,2
5. http://beej.us/guide/bgipc/html/single/bgipc.html

Nice tutorials on RPC and shared Memory:
(1) Tutorial on RPC
(2) RGPGen (and 2nd link similar to Dave's tutorial).
(3) Shared Memory
(4) Linux journal tutorial that uses avg.x
(5) Beej's Guide to PIC

See Textbook i.e., reading assignment

Remote Method Invocation

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
- RMI allows a Java program on one machine to invoke a method on a remote object.
- Possible to Pass Objects(remote, local) as parameters to remote methods (via serialization).



Marshalling Parameters

- Client invoke method: someMethod on a remote object Server