Remote Procedure Calls

Dan Savel, dxs221
EECS 338, Spring 2011

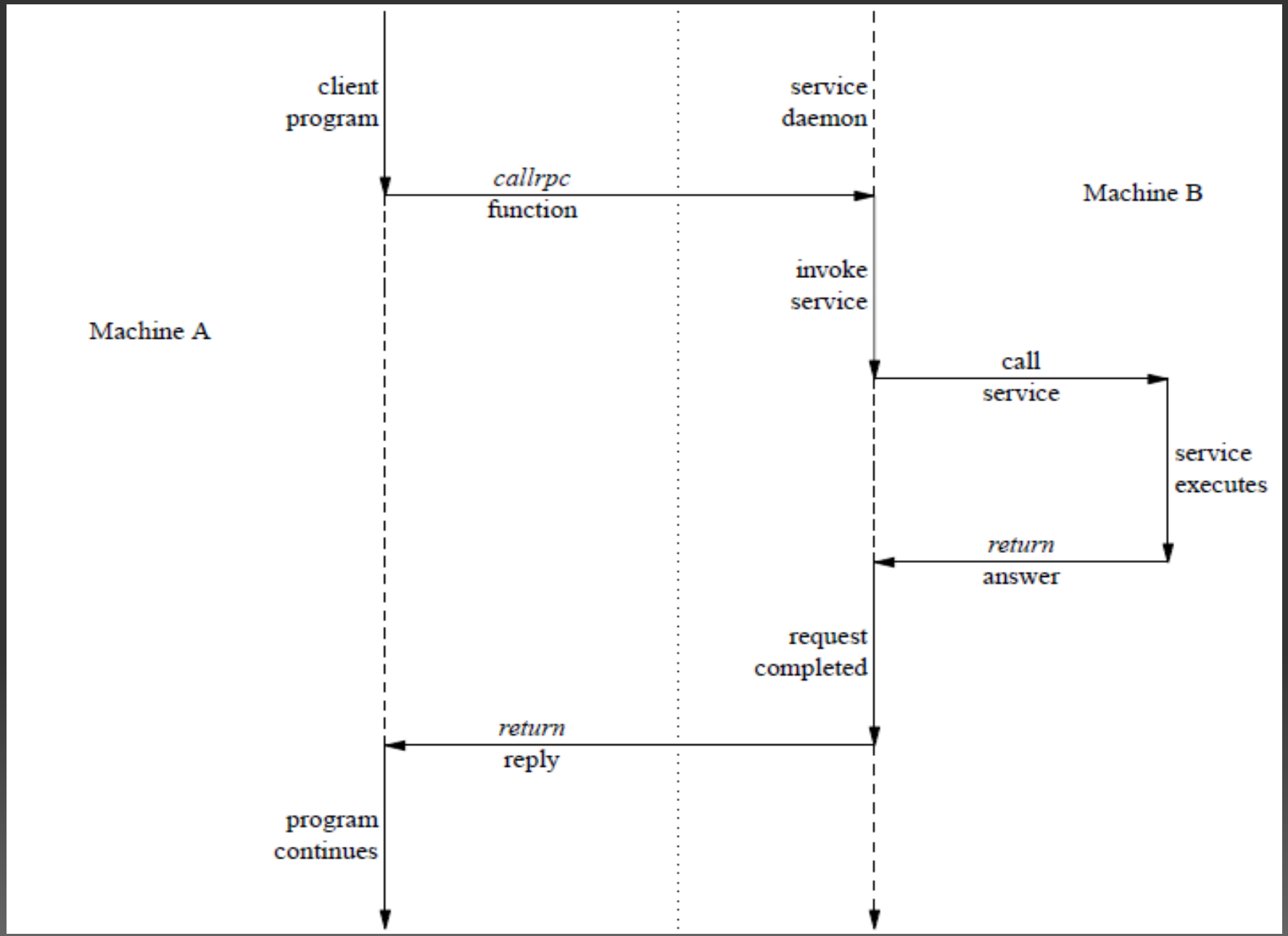
What is a Remote Procedure Call?

- Executing a procedure in an application running on a remote server
- Transport Protocol and Serialization are abstracted away to work transparently

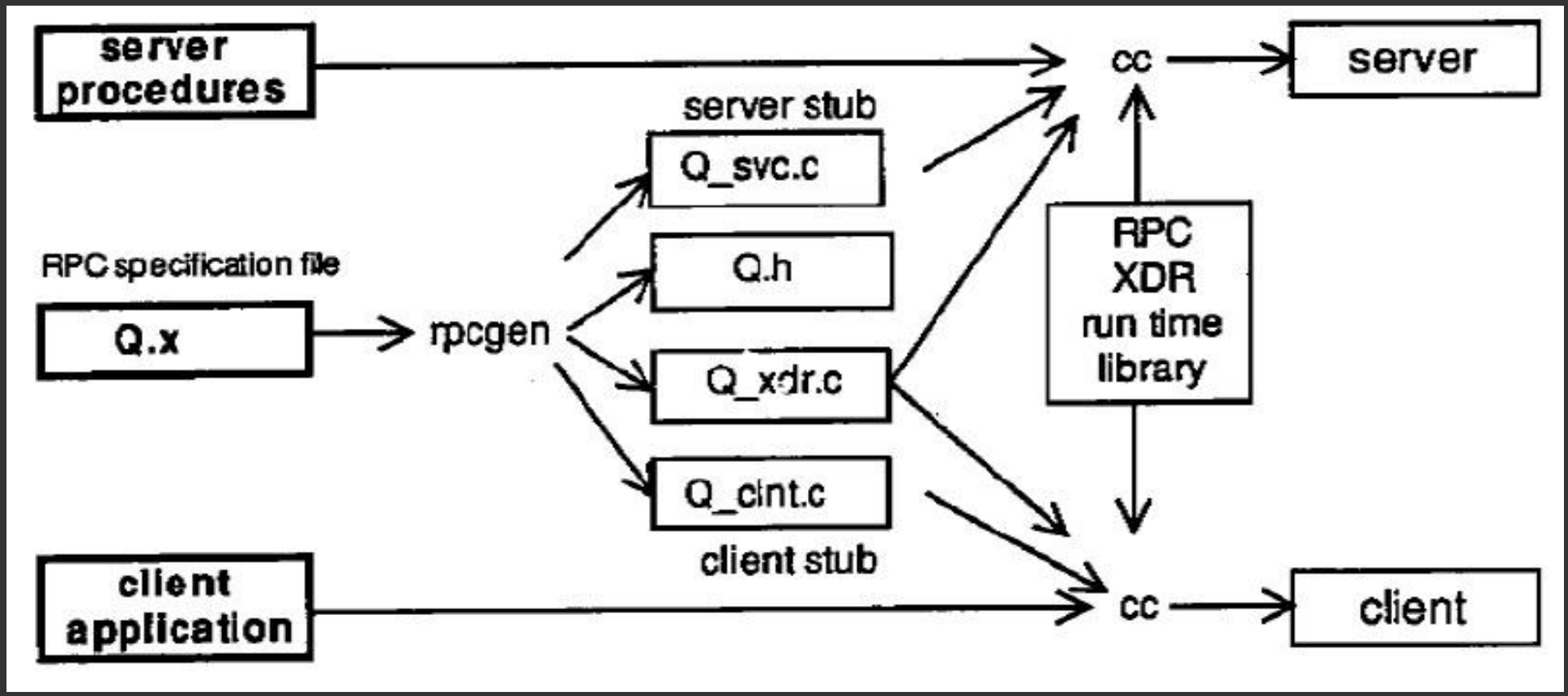
What happens when you call an RPC?

1. A client applications calls an RPC method
2. The client stub code serializes the data and sends it to the server
3. The server stub code will receive the serialized data
4. The data is un-serialized and put into a usable format
5. Server application processes the request
6. The server stub code serializes the response and sends it back to the client
7. Client stub code decodes the response
8. Client processes response

RPC Timeline



So What Do I Make?



Files that are made by the user. i.e. Q.x, server.c, and client.c

Files that are created when rpcgen is run on the .x file

Applications that are made after compiling all the right parts together

What Do I Make? Cont.

- User will create at least 3 source code files, a .x file, a client in c, and a server in c.
- Running rpcgen on the .x file will create a handful of files, so if the file was Q.x the files that would be generated are...
  - Q.h
  - Q_clnt.c
  - Q_svc.c
  - Q_xdr.c
- Do Not Modify these files

What are those created files?

- Q.h
  - Interface which needs to be included in all source files which will be using these RPC Procedures
  - Links the source files together
- Q_xdr.c
  - Where calls for XDR conversion are defined
- Q_clnt.c
  - Stub file to be used with client applications
- Q_svc.c
  - This is the server application, contains the main method

So... Making that .x file

- Has its own language, the XDR/RPC Language
  - VIM will highlight keywords for this
  - emacs doesn't

- Define data types, programs, versions, and procedures
- Most importantly is to define the program, version, and then procedure

Defining a simple procedure

- This is what is contained in file 'test.x'

```
program TEST_PROG
{
    version TEST_VERS
    {
        int TEST_PROC(int) = 1;
    } = 1;
} = 0x21140000
```

- After running 'rpcgen test.x' there will only be 3 created files
  - test.h, test_clnt.c, test_svc.c
  - no test_xdr.c since I did not define any structs or types

Writing a Server, server.c

```
#include "test.h"

int *test_proc_1_svc(int *in, struct svc_req *rqstp)
{
    static int response;
    response = *in;
    printf("Server Received %d and sent %d\n", *in, response);
    fflush(NULL);
    return(&response);
}

//Gives the body of the procedure defined in the .x file
```

Writing a Client, client.c

```
#include "test.h"

int main(int argc, char**argv)
{
    CLIENT *cl;
    if(argc != 3) return 1;
    cl = clnt_create(argv[1], TEST_PROG, TEST_VERS, "udp");
    int i = atoi(argv[2]);
    int* result = test_proc_1(&i, cl);
    status = *result;

    printf("Received result %d\n", *result);
    clnt_destroy(cl);
    return 0;
}
```

For Assignment 7

- Files to make
  - Cookie.x
    - XDR/RPC file
  - Tina.c
    - A client application which calls an RPC
  - Judy.c
    - A client application which calls an RPC
  - Mother.c
    - A server procedure definition file
    - Contains the body for the RPC

Compiling a Client

- Compile all necessary components individually
  - gcc -c Cookie_xdr.c
  - gcc -c Cookie_clnt.c
  - gcc -c Tina.c
  - gcc -o Tina Tina.o Cookie_xdr.o Cookie_clnt.o
- Can shove all these commands into a makefile
- Same for Judy.c

Compiling a Server

- Compile all necessary components individually
  - gcc -c Cookie_xdr.c
  - gcc -c Cookie_svc.c
  - gcc -c Mother.c
  - gcc -o Mother Mother.o Cookie_xdr.o Cookie_svc.o
- Or likewise put it into a makefile

Terminating the Server

- In the assignment, all processes are to terminate
- On its own the RPC server application will just continue to run
- Suggested Course of Action would be to create a specific RPC procedure which is an exit command

Updated test.x file

```
program TEST_PROG
{
    version TEST_VERS
    {
        int TEST_PROC(int) = 1;
        void TEST_EXIT(int) = 2;
    } = 1;
} = 0x21140000
```

- Now there are two methods defined

Updated server.c file

```
#include "test.h"

...

void *test_ext_1_svc(int *in, struct svc_req *rqstp)
{
    printf("Request for Termination Received\n");
    fflush(NULL);
    exit(0);
}
```

- Adding the body of the second method

Updated client.c file

```
#include "test.h"

int main(int argc, char**argv)
{
    ...
    if (i == -1)
    {
        test_exit_1(&i, cl);
        printf("Terminated Server\n");
        return 0;
    }
    ...
}
```

Nuances of Terminating Server

- The server termination call can't both return a value and tell the server to exit.
  - Once exit() is called the process is terminated so no more actions will be executed
  - Once return is called no further instructions in that method will be executed
  - So only one or the other can really be done.

- So the call to this function won't return to the client properly
  - It will timeout or error out, typically returning NULL to indicate this
  - This is probably fine though since the server is no longer necessary at this point

RPC Wrapup

- Questions about RPC?
- Short demo of exit working
- Remote Communication without using RPC
  - Socket Programming