Network Programming

Samuli Sorvakko/Nixu Oy

Telecommunications software and Multimedia Laboratory
T-110.4100 Computer Networks

October 16, 2008

Agenda

Introduction

Introduction and Overview	Socket Programming	Higher-level interfaces	Final thoughts
○●○	○○○○○○○	○○○○○○○○○	○○○○○○○

Introduction

Overview

- Wide-area concurrency
- Two or more entities
  - Client-server, peer-to-peer, unidirectional or bidirectional multicast, broadcast, ...
- Multiple levels of information exchange
  - From TCP/IP point of view, HTTP is an application
  - From SOAP or AJAX point of view, HTTP is a transport
  - From a suitably abstracted framework's point of view, SOAP is a transport...
- All quite complex, eh?

Introduction and Overview	Socket Programming	Higher-level interfaces	Final thoughts
○○●	○○○○○○○	○○○○○○○○○	○○○○○○○

Introduction

Managing complexity

- Well-known protocols
- Layering
- Modularization / compartmentalization

Socket Programming

| Introduction and Overview | Socket Programming | Higher-level interfaces | Final thoughts |
| 000 | ●000000 | 000000000 | 0000000 |

Overview

Overview

- The UNIX way
- Introduced in 1983 (4.2 BSD Unix)
- Bind together software and the communication channels they use

Overview cont'd.

- Bind together four items:
    - Remote host address
    - Remote host port number
    - Local host address
    - Local host port number
- Also additional information:
    - Socket protocol (Local, IPv4, IPv6, IPX, X25, ...)
    - Communication type (Stream, datagram, raw, ...)
    - Other options (blocking/non-blocking, keepalive, ...)

Introduction and Overview	Socket Programming	Higher-level interfaces	Final thoughts
ooo	oooooo	oooooooo	ooooooo

Client sockets

Client sockets

- Create a socket (binding it to a file descriptor)
- Connect the socket with the other party

```
int sockfd=socket(PF_INET, SOCK_STREAM, 0);
connect(sockfd,
        (struct sockaddr *) &remoteaddr,
        sizeof(struct sockaddr));
```

| Introduction and Overview | Socket Programming | Higher-level interfaces | Final thoughts |
| 000 | 0000000 | 000000000 | 0000000 |

Client sockets

Client sockets cont'd.

- Of course need to verify return values
- The remoteaddr struct needs to be filled
  - sin_family (AF_INET)
  - sin_port (generally via htons())
  - sin_addr (usually from hostent struct from gethostbyname())

Introduction and Overview	Socket Programming	Higher-level interfaces	Final thoughts
000	0000●00	000000000	0000000

Server sockets

Server sockets

- A bit more complicated than the client
- Again, socket needs to be created
- Then bound to desired protocol, port and listening address
- After that, indicate willingness to listen to the OS
- Now ready to accept connections

```
int sockfd=socket(PF_INET, SOCK_STREAM, 0);
bind(sockfd,
        (struct sockaddr *) &myaddr,
        sizeof(struct sockaddr));
listen(sockfd, backlog);

sin_size=sizeof(struct sockaddr_in);
incoming_fd=accept(sockfd,
                    (struct sockaddr *)&remote_addr,
                    &sin_size);
```

Server sockets cont'd.

- What is usually done here is to fork() a child process
- New connections can be accepted as quickly as possible
- Old connections are served by the childs asynchronously
- Other keywords: select(2), poll(2)

Sockets recap

- Examples were for TCP sockets, UDP similar
- Very simplified examples, don't do it like this :)
- What is sent over the socket is decided by programmer
- Actual communication is handled by OS, socket operations are syscalls

Higher-level interfaces

Introduction and Overview	Socket Programming	Higher-level interfaces	Final thoughts
000	0000000	●00000000	0000000

RPC

Remote Procedure Call

- Developed by Sun Microsystems
- Originally for NIS and NFS
- Defines a data representation for binary information (byte orders!)

Introduction and Overview	Socket Programming	Higher-level interfaces	Final thoughts
000	0000000	0●0000000	0000000

RPC

Remote Procedure Call cont'd.

- Uses a portmapper `portmap/rpcbind` instead of direct communication
- RPC server opens up a free UDP or TCP port and registers with portmapper
- RPC client contacts portmapper and gets exact location of server
- Also contains some options for authentication etc.

Introduction and Overview	Socket Programming	Higher-level interfaces	Final thoughts
000	0000000	000●00000	0000000

RMI

Java Remote Method Invocation

- Also developed by Sun Microsystems
- Provides a way for Java object invocation from other Java VMs
- Supports object serialization

Java Remote Method Invocation cont'd

- Remote end:
  - Export interfaces
    `public interface MyInterface extends Remote{}`
  - Comms failures will be reported with `RemoteException`
  - Creates instance(s) of a remote object
  - Register the object(s) with RMI remote object registry
- Local end:
  - Request the object from the remote server, which returns a "Stub" instance
  - Methods invoked on the stub are run on the server, with RMI serializing and deserializing the communication

CORBA

- Common Object Request Broker Architecture
- Vendor-independent way for remote objects
- Specified by Object Management Group (OMG...)
- IDL, Interface Definition Language describes exported interfaces
- Similar to RMI in principle
- Mappings exist for C, C++, Java, COBOL, Lisp, Python...

CORBA cont'd

- Interface is well separated from the implementation
- CORBA is well suited for middleware ("glue") tasks
- Allows for access control on object level

| Introduction and Overview | Socket Programming | **Higher-level interfaces** | Final thoughts |
| 000 | 0000000 | 000000●00 | 0000000 |

DCOM

Microsoft's offerings

- Distributed Component Object Model (DCOM)
- Based on "local" COM, with added RPC, serializing and garbage collection functionality

- .NET Remoting
- Part of the .NET framework

- Windows Communication Foundation
- Unifies .NET comms programming models
  - Web services, .NET Remoting, Message Queues, Distributed Transactions
  - Can also serve AJAX web request via JSON encoder

Idea here is exactly the same as in CORBA et al, remote invocation of procedures or methods in objects.

| Introduction and Overview | Socket Programming | Higher-level interfaces | Final thoughts |
| 000 | 0000000 | 000000000 | 0000000 |

Webserv

Web Services

- "Leverage the power of the Web"
- Machine-to-machine communication
- SOAP: Extensible, XML-based communication over HTTP
- WSDL: Interface description language
- UDDI (Universal Description Discovery and integration): Publishing and discovery of Web services
- Can be used in many ways; RPC emulation, "Service-oriented architecture" (SOA), Representational State Transfer (REST)

| Introduction and Overview | Socket Programming | Higher-level interfaces | Final thoughts |
| 000 | 0000000 | 00000000● | 0000000 |

Webserv

Web Services

- AJAX (Asynchronous JavaScript and XML) could also be categorized as a web service
- Not strictly machine-to-machine
  - User's browser may do operations without interaction
- Data exchange between server and browser
- Only a part of the web page is refreshed
- Communication with XMLHttpRequests (or IFrames)
- Not a standard or a technology, describes functionality

Final thoughts

Security

- Cannot trust the network
- Client cannot trust server
- Server *must not* trust client

Security - Input handling

- Being on a network means communicating with more entities than you might think
- What if one of the entities is malicious?
- What happens to a server if a client sends it e.g. \0's, SQL statements, very large amounts of data...

Security - Input handling

- Usually there are limits for things
  - Field length, allowed characters, timeouts etc.
- It is best to make the limits explicit and force validation
- Example: A field in a text-based protocol contains a length for the payload (e.g. HTTP Content-Length:)
  - Check that the length is not negative
  - Check that the length is a number
  - Do *not* trust the reported length...
- Example: A server-side AJAX handler will look up entries from an SQL database
  - Check that the request is sane (e.g. discard SQL wildcards)
  - Check that the request contains NO fragments of SQL statements
  - Remember to check for different character encodings, character entities etc...
- Input handling should be handled in a consistent manner throughout the application

Introduction and Overview	Socket Programming	Higher-level interfaces	**Final thoughts**
000	0000000	000000000	0000●000

Security

Security - Application logic

- Usually apps have different states they can be in
- Waiting for connection, authenticating, authorized but idle, data transfer....
- States can be implicit or explicit
- As with input handling, explicit usually better
- Need to verify that the state transition is proper
    - Initiating a monetary transaction not allowed without authentication and authorization
    - Inserting routing table entries not allowed if routing table static
    - ...
- States are application specific
- State machines will help immensely (don't we all love theoretical computer science :)

Security - yet again...

- Use ready and tested protocol implementations if possible
- Use well-known protocols if possible
- Design protocols with security on mind from the start
- Always test for robustness, not only compliance

Further reading

- Richard Stevens: UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998, ISBN 0-13-490012-X
- `man 2 socket`, `man 2 connect`, `man 2 bind` and other UNIX man pages
- Sun Java RMI guides, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/
- Object Management Group CORBA FAQ and other documentation, http://www.omg.org/gettingstarted/corbafaq.htm
- Secure Programming for Linux and Unix HOWTO, http://www.dwheeler.com/secure-programs/

Discussion

Comments? Remarks? Questions?