Systems Architecture

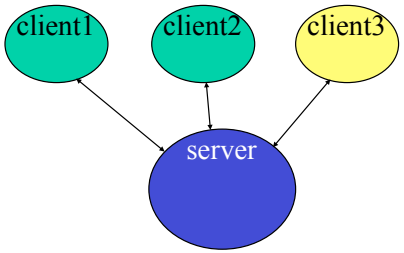
Client-Server Systems
(Week 11 hour 1)

Is the architecture?

- We continue our smorgasbord of alternative system structures.
  - A little like our buffet of patterns.
- Last week we discussed a monolithic approach
  - No good when users need to share the data managed by the system to work collaboratively.
- This week we describe a few optional ways of building client server systems.
  - Client server
  - "Raw" Sockets
  - Distributed Objects
  - Enterprise Java Beans
- Just a quick tour to introduce a few concepts.
  - Much (much) complexity beyond our scope.

Client/Server

- In general, any application where multiple clients connect to a single server.



- one client program (most typical)
  or
- multiple client programs

IPC

- "Inter-Process Communications"
  - How processes will communicate and synchronize with one-another.
  - communications mechanisms:
    - shared memory
      - very fast
      - can't use over a network
        - » well, you can
    - message passing
      - can use over a network
      - slower
        - » well, not always
  - will consider only message passing (much more common programming model at present)

IPC Protocols

- Basic message-passing mechanisms provide for a byte-stream only.
- Must implement various protocols on top of this
  - sockets
  - RPC (remote procedure call)
  - DO (distributed objects)

Java Sockets code example

```
public class Server {
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(1234);
        Socket client = server.accept();
        BufferedReader fromClient = new BufferedReader(
                        new InputStreamReader(client.getInputStream()));
        System.out.println(fromClient.readLine());
    }
}
```

```
public class Client {
    public static void main(String[] args) throws Exception {
        Socket server = new Socket("localhost", 1234);
        DataOutputStream toServer = new DataOutputStream(
                                            server.getOutputStream());
        toServer.writeBytes("hello server");
        server.close();
    }
}
```


Performance

- Latency
  - The time to go back and forth
- Bandwidth
  - The amount of data that can be sent
- Analogy to dump truck of disks
  - Can carry quarter million or so (about 1 Gbyte) DVD's
    - I reckon in the region of 10,000 per cubic m
  - 5 hours from Montreal to Toronto (18,000 sec)
  - Better than 10G/sec.
  - Good bandwidth. Crummy latency.

Test System

- Windows 2000 Java Server
  - Network
    - 100 Mbit/s ethernet
  - CPU
    - dual 1GHz processors
  - Memory
    - 1 GByte
- Windows 98 Java Client
  - Network
    - 100 Mbit/s ethernet
  - CPU
    - 366 MHz
  - Memory
    - 96 MByte

Java/Windows Performance Measures

- Latency: Sending "hello server\r\n" back and forth
  - Local method calls
    - .13 usec/2call
    - 100 cycles or so..
  - Socket on local machine
    - 70 usec / 2call (x500 i.e 500 times slower than local method call)
  - Socket on remote machine
    - 320,000 usec /2call (x5,000 , x2,500,000)
- Bandwidth
  - Sending "hello server\r\n" to server repeatedly
    - 1400 usec / 2call (x10,000 , x230)

Performance

	In Process	Network
Latency	1	2,500,000
Bandwidth	1	10,000

RPC 250x slower than local call

C/Windows Performance Measures

- Latency: Sending "hello server\r\n" back and forth
  - Local method calls
    - .01 usec/2call (10x Java i.e. 10 times faster than Java)
    - Only 10 cycles. Not copying string. Just passing it by reference.
  - Socket on local machine
    - 12 usec / 2call (6x Java)
  - Socket on remote machine
    - 840 usec /2call (380x Java)

Performance

	In Process	Network
Latency	1	84,000

Performance Implications

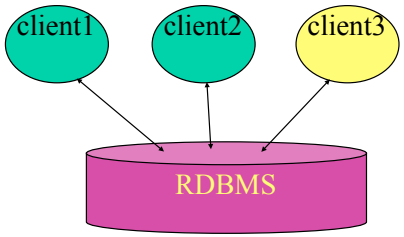
- Modern research micro-kernel operating systems like (UofT and IBM's) k42 require about one thousand cycles for RPC from one local process to another..
  - even C/Windows is pretty slow compared to what it could be..
- Meanwhile, back where the apps are being built..
- Do as few calls as possible over the net
  - "Slice" app up correctly or performance will be terrible.
- Consider asynchronous approaches?
  - problem: success/failure indications
  - send lots of stuff, then synchronize
- Use bigger transactions
- Prefer one call with lots of data to many calls with the same amount of data
  - but not by much
- Send as little data as possible

Relational Databases

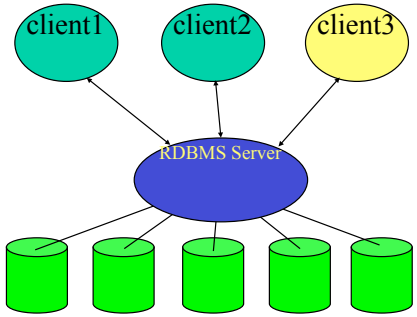
- Most common type of client/server software is where the server is an RDBMS server:
  - Oracle
  - SQLserver
  - Sybase
  - Informix
  - mySQL
  - postgress

Relational Databases

- Most common client/server program is where the server is a relational database server.
  - warning: some use the term client/server to refer to this usage exclusively (we won't).



Relational Database Implementation



Proprietary Client/Server

- Most of the time, when are assisted by a clerk in government or business plugging data into a graphical user form, they are operating a proprietary client.
  - Powerbuilder was dominant at one point.
  - Microsoft visual basic apps query databases using proprietary ODBC protocol.
  - Popularity somewhat eclipsed by "thin client" applications.
  - All big database vendors have their own equivalent products.
  - Usually include elaborate screen builder development environment
  - Often include own language to control client.
  - Include very elaborate widgets that do things like populate tabular data from the results of database queries.
  - Total victory of tactics (speed of development) over strategy (reusable and well packaged business logic).

Database Access

- Access using SQL (Standard Query Language)

```
select itemname,quantity
    from
        orderitems,items
    where
        orderid = 239
            and
        orderitems.itemid = items.itemid
```

includes notion of a "stored procedure" if a chunk of SQL like this is parameterized and named

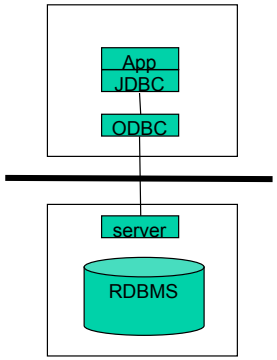
query result	
itemname	quantity
bread	2142
sugar	345

Programmatic Database Access

- Can access database by
  - typing commands at an sql command prompt
  - by running a GUI tool
  - programmatically
    - ODBC
      - Open Database Connectivity – Microsoft standard API
      - ANSI/ISO CLI is ODBC level1 compliant (Call Level Interface)
        - » (see also DAO, OLE DB and ADO)
    - JDBC
      - very similar to ODBC
    - Various embedded SQL hacks

JDBC

- All sorts of possible configurations of client-side & server-side drivers



Database Access from Java

```
import java.sql.*;

    public class Main {
        private static final query =
        “select itemname,quantity “ +
        “from orderitems,items “ +
        “where orderid=1 and orderitems.itemid=items.itemid”;

        public static void main(String[] args) throws Exception {
            Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
            Connection c = DriverManager.getConnection(“jdbc:odbc:grocery”);
            Statement s = c.createStatement();
            if( s.execute(query) ) {
                ResultSet r = s.getResultSet();
                printResults(r);
            }
        }

        private static void printResults(ResultSet r) throws Exception {
            final int nC = printHeadings(r);
            printRows(nC, r);
        }
        …
    }
```


Database Access from Java

```
private static int printHeadings(ResultSet r)
throws Exception {

    ResultSetMetaData m = r.getMetaData();
    final int nC = m.getColumnCount();
    for(int c = 1; c <= nC; c++) {
        System.out.print(m.getColumnName(c));
        System.out.print(“\t”);
    }
    System.out.println();
    return nC;
}
```


Database Access from Java

```
private static void printRows(int nC, ResultSet r)
throws Exception {
    while( r.next() ) {
        for(int c = 1; c <= nC; c++) {
            System.out.print(r.getString(c));
            System.out.print(“\t”);
        }
        System.out.println();
    }
}
```


Without ODBC

```
Class.forName(
    “org.gjt.mm.mysql.Driver“
    );



Connection c = DriverManager.getConnection(
    "jdbc:mysql://penny.dhcp.cs.toronto.edu/grocery”
    );
```

Performance

- localhost
  - JDBC:ODBC
    - 850 us/query
  - JDBC:MYSQL
    - 500 us/query
- over network
  - JDBC:ODBC
    - 3,800 us/query
  - JDBC:MYSQL
    - 1,600 us/query

• local Java method call
– 0.13 us/query
• C socket over network
• 840 us/query

database does work.

Data Compatibility

- Issue with any sort of system is how to support changes in data format from release to release of the software:
  - backwards compatible
    - newer releases of the software can open older datasets
  - forwards compatible
    - older releases of the software can open newer datasets
- General approach
  - have some sort of flexible header format
  - for backwards compatibility:
    - encode a current data version number
  - for forwards compatibility
    - store the oldest data version number such that
      - older software that uses that data version can still use this data

RDBMS Compatibility Advantages

- RDBMS's have 2 advantages w.r.t compatibility:
  - The data is not fragile.
    - e.g., in a binary file format, one small change somewhere can screw up the whole file
    - in SQL the schema can change considerably yet the data can still be accessed
  - Query engines can be very fast. Optimizers are sophisticated.
  - RDBMS are intended to be the main store for data as applications are developed and mature.
    - And languish on life support interminably..
  - RDBMSs support schema evolution
    - SQL
      - CREATE TABLE
      - MODIFY TABLE

        Can work on in-place databases

UpdateDatabase Code Example

```
private void updateDatabase() {
    int version = getDataVersion();
    if( version < 1 )
        die("DB consistency error: Version number must be 1 or greater");
    switch(version) {
        case 1:
            updateDatabaseToVersion2();
            // fall-through
        case 2:
            updateDatabaseToVersion3();
            // fall-through
        case 3:
            o.println("<Database is up-to-date>");
            break;
        default:
            die("Database was created with newer version of software");
            break;
    }
}
```

Update Database Code

```
private void updateDatabaseToVersion2() {
    o.println("<Converting database from version 1 to version 2>");
    try {
        sqlup("ALTER TABLE Coders ADD COLUMN w REAL");
        sqlup("UPDATE Coders SET w = 0.6");
        sqlup("UPDATE Version SET version = 2");
        sqlcommit();
    } catch(Exception e) {
        try {
            sqlrollback();
        } catch(Exception e2) { }
        die("Error converting database to version 2: " +
              e.getMessage());
        }
    }
```