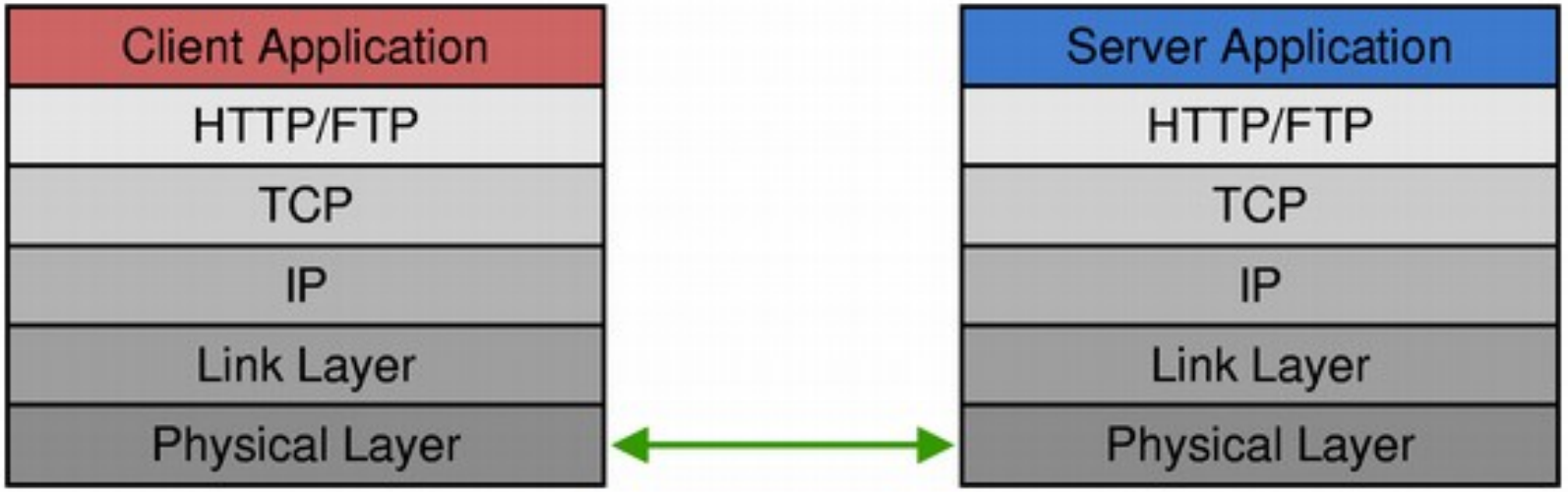
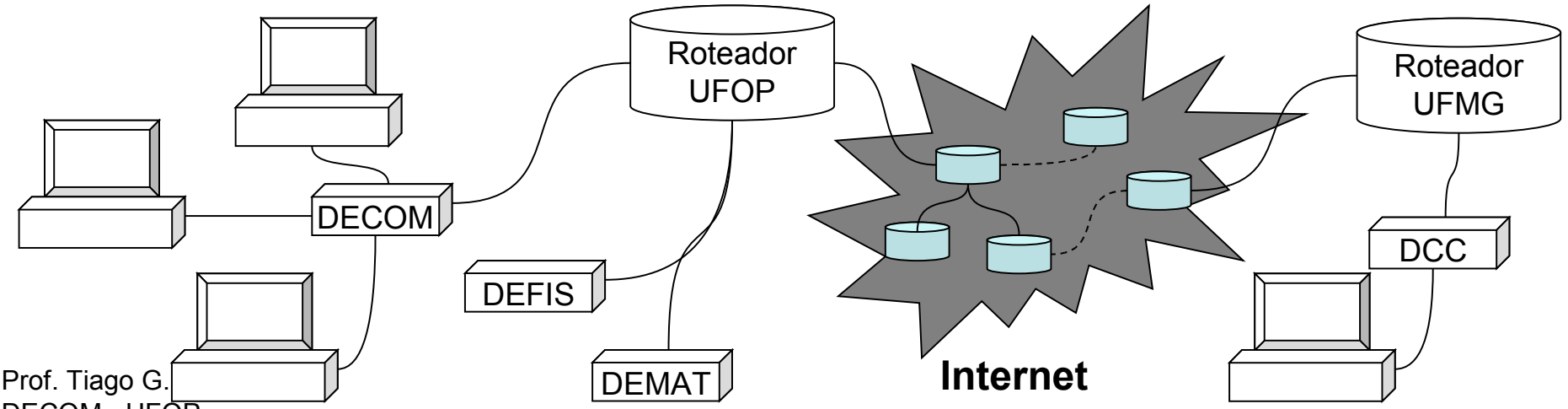
Network Programming in Qt

Prof. Tiago Garcia de Senna Carneiro
DECOM – UFOP
2007

TCP/IP Protocol Stack

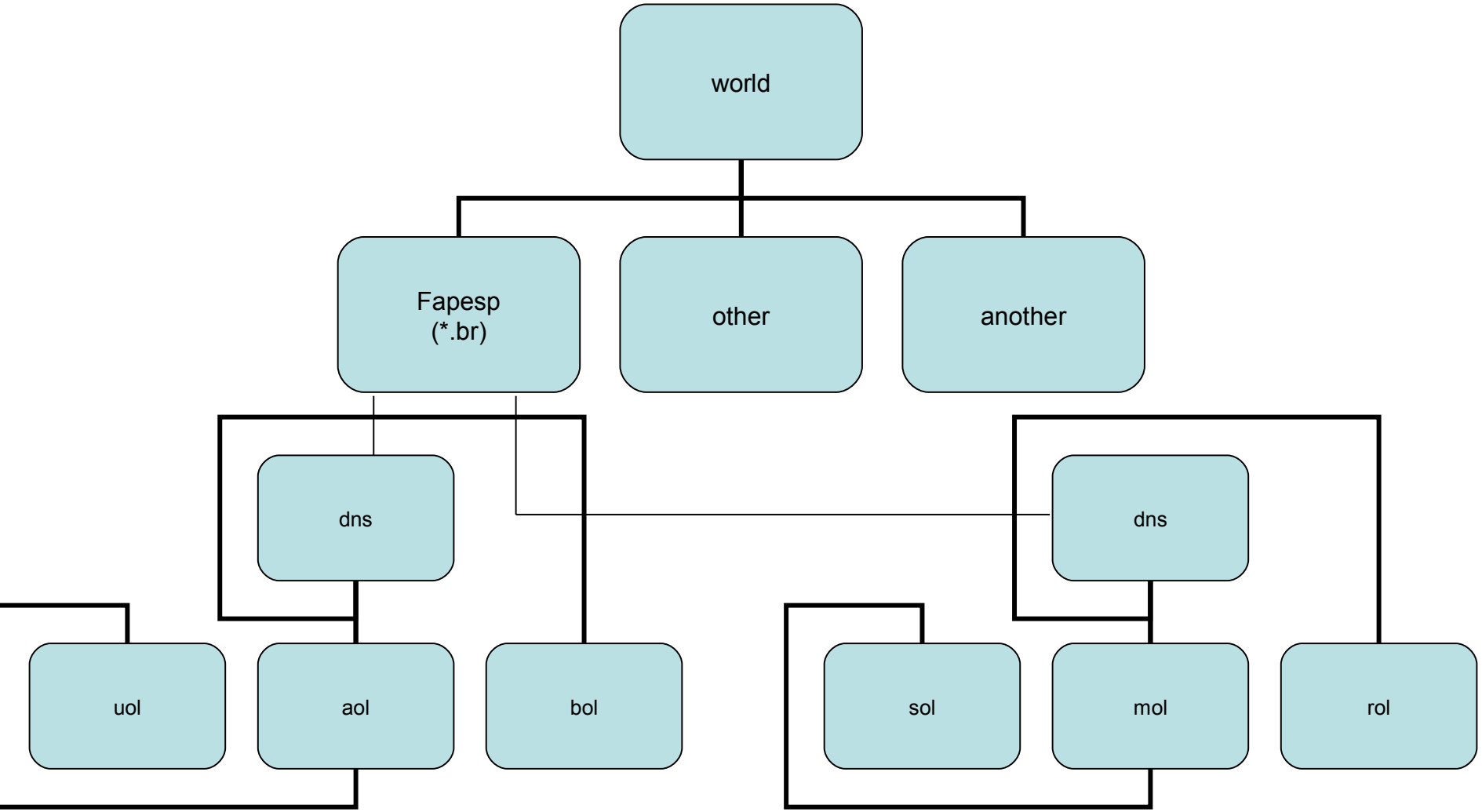




Domain Name Server (DNS)

- A maioria dos seres humanos conseguem memorizar seqüências de até sete números
- DNS associa endereços IP a hostname's para simplificar a memorização do ser humano
- Note que um endereço é diferente de um nome de host
  - As pessoas chamam hostname's de endereço... Nesse curso será feita uma clara (e necessária) distinção

Domain Name Server (DNS)



Network Classes

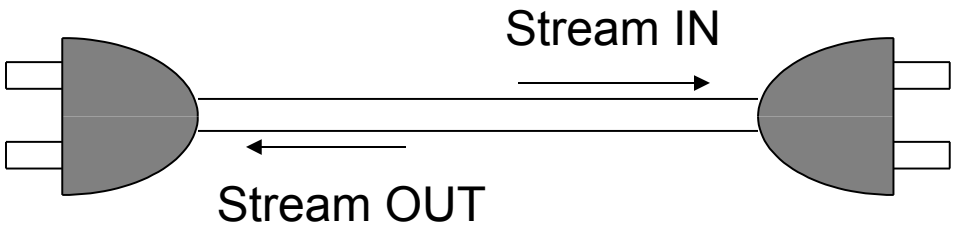
QAbstractSocket	The base functionality common to all socket types
QFtp	Implementation of the client side of FTP protocol
QHostAddress	IP address
QHostInfo	Static functions for host name lookups
QHttp	Implementation of the HTTP protocol
QHttpHeader	Header information for HTTP
QHttpRequestHeader	Request header information for HTTP
QHttpResponseHeader	Response header information for HTTP
QNetworkAddressEntry	Stores one IP address supported by a network interface, along with its associated netmask and broadcast address
QNetworkInterface	Listing of the host's IP addresses and network interfaces
QNetworkProxy	Network layer proxy
QTcpServer	TCP-based server
QTcpSocket	TCP socket
QUdpSocket	UDP socket
QUrlInfo	Stores information about URLs

java.net.InetAddress

- Representa um endereço IP
- Pode ter 32 ou 128 bits
  - Veja as subclasses: Inet4Address e Inet6Address
- Endereços da forma: ddd.ddd.ddd.ddd
  - Usando o padrão atual: IPv4
  - Quadra pontuada (dotted quad)

Soquetes

- Abstração que permite ao programador tratar uma conexão de rede como um stream de dados.
- Libera do programador dos detalhes de baixo nível da rede
- Um socket conecta dois *hosts*



Portas notáveis

- De 1 a 1023
- Breve lista
  - 07 echo
  - 13 daytime
  - 21 FTP
  - 23 Telnet
  - 25 SMTP
  - 80 HTTP
  - 110 POP3
  - 1099 RMI Registry
- Lista detalhada em:

  http://pt.wikipedia.org/wiki/Lista_de_portas_de_protocolos

Operações Básicas dos Sockets

- Conectar a máquinas remotas
- Enviar dados
- Receber dados
- Encerrar conexão

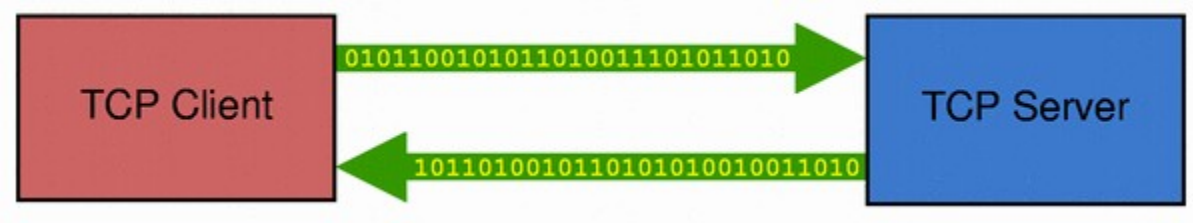
Cliente

- Conectar-se a portas lógicas
- Aguardar chegada de dados
- Aceitar conexão de máquinas remotas

Servidor

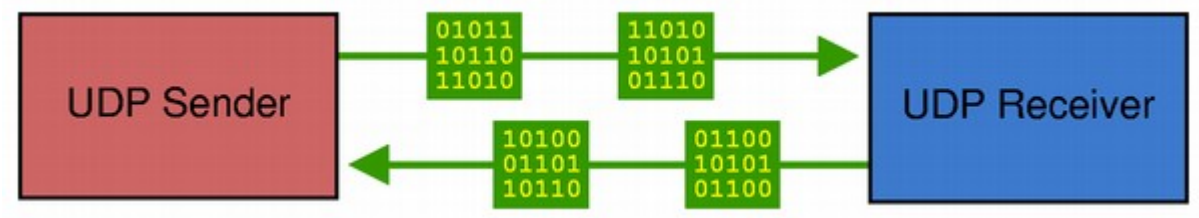
Transmission Control Protocol

- TCP Protocol:
  - Reliable
  - Stream-oriented
  - Connection-oriented

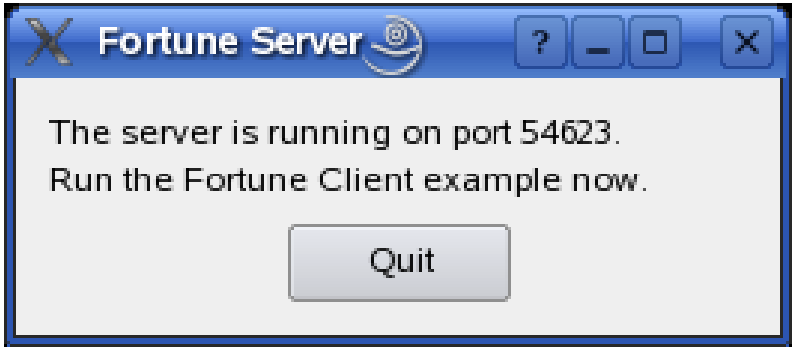


User Datagram Protocol

- UDP Protocol:
  - Lightweight
  - Unreliable
  - Datagram-oriented
  - Connectionless



TCP Single **Server** Application




```
#include <QDialog>

class QLabel;
class QPushButton;
class QTcpServer;

class Server : public QDialog {
        Q_OBJECT
public:
        Server(QWidget *parent = 0);
private slots:
        void sendFortune();
private:
        QLabel *statusLabel;
        QPushButton *quitButton;
        QTcpServer *tcpServer;
        QStringList fortunes;
};
```


Server.h

TCP Single **Server** Application

```
Server::Server(QWidget *parent)  : QDialog(parent)
{
    statusLabel = new QLabel;
    quitButton = new QPushButton(tr("Quit"));
    quitButton->setAutoDefault(false);

    tcpServer = new QTcpServer(this);
    if (!tcpServer->listen()) {
        QMessageBox::critical(this, tr("Fortune Server"),
                              tr("Unable to start the server: %1.")
                              .arg(tcpServer->errorString()));
        close();
        return;
    }
  ...
```


Server.cpp

TCP Single **Server** Application

```
...
statusLabel->setText(tr("The server is running on port %1.\n"
                        "Run the Fortune Client example now.")
                     .arg(tcpServer->serverPort()));

  fortunes << tr("You've been leading a dog's life. Stay off the furniture.")
           << tr("You've got to think about tomorrow.")
           << tr("You will be surprised by a loud noise.")
           << tr("You will feel hungry again in another hour.")
           << tr("You might have mail.")
           << tr("You cannot kill time without injuring eternity.")
           << tr("Computers are not intelligent. They only think they are.");

  connect(quitButton, SIGNAL(clicked()), this, SLOT(close()));
  connect(tcpServer, SIGNAL(newConnection()), this, SLOT(sendFortune()));
  ...
```


Server.cpp

TCP Single **Server** Application

```
    ...
    QHBoxLayout *buttonLayout = new QHBoxLayout;
    buttonLayout->addStretch(1);
    buttonLayout->addWidget(quitButton);
    buttonLayout->addStretch(1);

    QVBoxLayout *mainLayout = new QVBoxLayout;
    mainLayout->addWidget(statusLabel);
    mainLayout->addLayout(buttonLayout);
    setLayout(mainLayout);

    setWindowTitle(tr("Fortune Server"));
}
```


Server.cpp

TCP Single **Server** Application

```
void Server::sendFortune() {

        QByteArray block;
        QDataStream out(&block, QIODevice::WriteOnly);
        out.setVersion(QDataStream::Qt_4_0);
        out << (quint16)0;
        out << fortunes.at(qrand() % fortunes.size());
        out.device()->seek(0);
        out << (quint16)(block.size() - sizeof(quint16));

        QTcpSocket *clientConnection = tcpServer->nextPendingConnection();
        connect(clientConnection, SIGNAL(disconnected()),
                                        clientConnection, SLOT(deleteLater()));
        clientConnection->write(block);
        clientConnection->disconnectFromHost();
}
```


Server.cpp

TCP Single **Server** Application

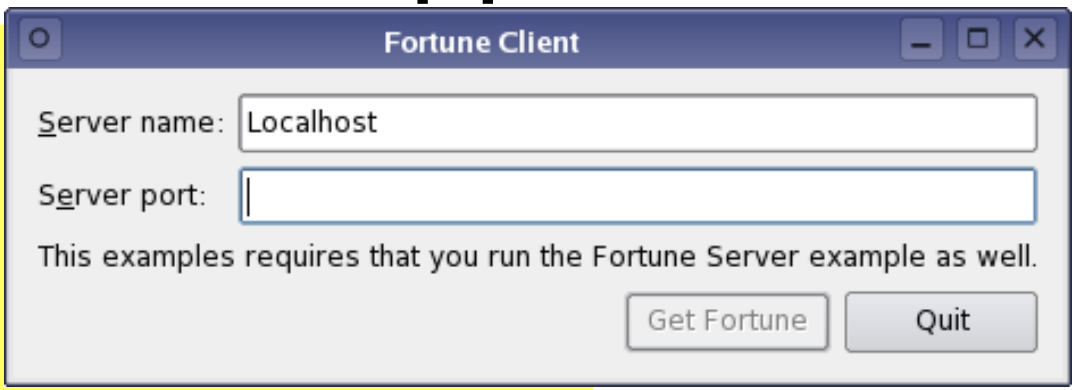
```
#include <QApplication> #include <QtCore>
#include <stdlib.h>
#include "server.h"

int main(int argc, char *argv[]) {
        QApplication app(argc, argv);
        Server server; server.show();
        qsrand(QTime(0,0,0).secsTo(QTime::currentTime()));
        return server.exec();
}
```


main.cpp

TCP Single **Client** Application

```
class Client : public QDialog {
        Q_OBJECT
public:
        Client(QWidget *parent = 0);
private slots:
        void requestNewFortune();
        void readFortune();
        void displayError(QAbstractSocket::SocketError socketError);
        void enableGetFortuneButton();
private:
        QLabel *hostLabel;
        QLabel *portLabel;
        QLineEdit *hostLineEdit;
        QLineEdit *portLineEdit;
        QLabel *statusLabel;
        QPushButton *getFortuneButton;
        QPushButton *quitButton;
        QDialogButtonBox *buttonBox;
        QTcpSocket *tcpSocket;
        QString currentFortune;
        quint16 blockSize;
};
```



Client.h

TCP Single **Client** Application

```
Client::Client(QWidget *parent) : QDialog(parent) {
        ...
        tcpSocket = new QTcpSocket(this);
        connect(hostLineEdit, SIGNAL(textChanged(const QString &)),
                this, SLOT(enableGetFortuneButton()));
        connect(portLineEdit, SIGNAL(textChanged(const QString &)),
                this, SLOT(enableGetFortuneButton()));
        connect(getFortuneButton, SIGNAL(clicked()), this, SLOT(requestNewFortune()));
        connect(quitButton, SIGNAL(clicked()), this, SLOT(close()));
        connect(tcpSocket, SIGNAL(readyRead()), this, SLOT(readFortune()));
        connect(tcpSocket, SIGNAL(error(QAbstractSocket::SocketError)),
                this, SLOT(displayError(QAbstractSocket::SocketError)));
        ...
}
```


Client.cpp

TCP Single **Client** Application

```
void Client::requestNewFortune() {
        getFortuneButton->setEnabled(false);
        blockSize = 0;
        tcpSocket->abort();
        tcpSocket->connectToHost(hostLineEdit->text(), portLineEdit->text().toInt());
}
```

```
void Client::enableGetFortuneButton() {
        getFortuneButton->setEnabled(!hostLineEdit->text().isEmpty() &&
                !portLineEdit->text().isEmpty());
}
```


Client.cpp

TCP Single **Client** Application

```
void Client::readFortune() {
        QDataStream in(tcpSocket);
        in.setVersion(QDataStream::Qt_4_0);
        if (blockSize == 0) {
                if (tcpSocket->bytesAvailable() < (int)sizeof(quint16)) return;
                in >> blockSize;
        }
        if (tcpSocket->bytesAvailable() < blockSize) return;
        QString nextFortune;
        in >> nextFortune;
        if (nextFortune == currentFortune) {
                QTimer::singleShot(0, this, SLOT(requestNewFortune()));
                return;
        }
        currentFortune = nextFortune;
        statusLabel->setText(currentFortune);
        getFortuneButton->setEnabled(true);
}
```

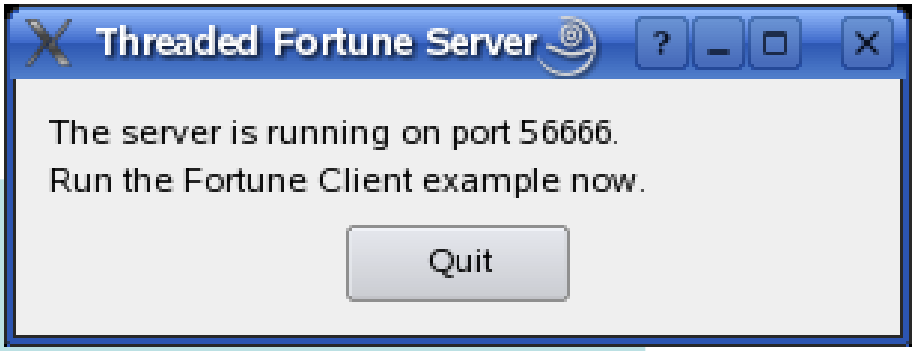

Client.cpp

TCP Single **Client** Application

```
void Client::displayError(QAbstractSocket::SocketError socketError) {
        switch (socketError) {
                case QAbstractSocket::RemoteHostClosedError: break;
                case QAbstractSocket::HostNotFoundError:
                        QMessageBox::information(this, tr("Fortune Client"),
                        tr("The host was not found. Please check the "
                          "host name and port settings."));
                break;
                case QAbstractSocket::ConnectionRefusedError:
                        QMessageBox::information(this, tr("Fortune Client"),
                        tr("The connection was refused by the peer. “
                         "Make sure the fortune server is running, “
                         "and check that the host name and port "
                        "settings are correct."));
                break;
                default:
                        QMessageBox::information(this, tr("Fortune Client"),
                        tr("The following error occurred: %1.") .
                        arg(tcpSocket->errorString()));
        }
        getFortuneButton->setEnabled(true);
}
```

main.cpp

TCP Threaded **Server** Application




```
#include <QStringList>
#include <QTcpServer>

class FortuneServer : public QTcpServer {
        Q_OBJECT
public:
        FortuneServer(QObject *parent = 0);
protected:
        void incomingConnection(int socketDescriptor);
private:
        QStringList fortunes;
};
```


FortuneServer.h

TCP Threaded **Server** Application

```
FortuneServer::FortuneServer(QObject *parent) : QTcpServer(parent) {
        fortunes << tr("You've been leading a dog's life. Stay off the furniture.")
                 << tr("You've got to think about tomorrow.")
                 << tr("You will be surprised by a loud noise.")
                 << tr("You will feel hungry again in another hour.")
                 << tr("You might have mail.")
                 << tr("You cannot kill time without injuring eternity.")
                 << tr("Computers are not intelligent. They only think they are.");
}
```

```
void FortuneServer::incomingConnection(int socketDescriptor) {
        QString fortune = fortunes.at(qrand() % fortunes.size());
        FortuneThread *thread = new FortuneThread(socketDescriptor, fortune, this);
        connect(thread, SIGNAL(finished()), thread, SLOT(deleteLater()));
        thread->start();
}
```


FortuneServer.cpp

TCP Threaded **Server** Application

```
#include <QThread>
#include <QTcpSocket>

class FortuneThread : public QThread {
        Q_OBJECT
public:
        FortuneThread(int socketDescriptor, const QString &fortune, QObject *parent);
        void run();
signals:
        void error(QTcpSocket::SocketError socketError);
private:
        int socketDescriptor;
        QString text;
};
```


FortuneThread.h

TCP Threaded **Server** Application

```
#include "fortunethread.h“
#include <QtNetwork>
```

```
FortuneThread::FortuneThread(int socketDescriptor, const QString &fortune, QObject *parent) :
        QThread(parent), socketDescriptor(socketDescriptor), text(fortune) { }
```

```
void FortuneThread::run() {
        QTcpSocket tcpSocket;
        if (!tcpSocket.setSocketDescriptor(socketDescriptor)) {
                emit error(tcpSocket.error());
                return;
        }
        QByteArray block;
        QDataStream out(&block, QIODevice::WriteOnly);
        out.setVersion(QDataStream::Qt_4_0);
        out << (quint16)0;
        out << text; out.device()->seek(0);
        out << (quint16)(block.size() - sizeof(quint16));
        tcpSocket.write(block);
        tcpSocket.disconnectFromHost();
        tcpSocket.waitForDisconnected();
}
```


FortuneThread.cpp

TCP Threaded **Server** Application

```
#include <QApplication>
#include <QtCore>
#include <stdlib.h>
#include "dialog.h"

int main(int argc, char *argv[]) {
        QApplication app(argc, argv);
        Dialog dialog;
        dialog.show();
        qsrand(QTime(0,0,0).secsTo(QTime::currentTime()));
        return dialog.exec();
}
```


Main.cpp

Building Network Applications

- The following declaration in a qmake project file ensures that an application is compiled and linked appropriately:
  - QT += network
- To include the definitions of the module's classes, use the following directive:
  - #include <QtNetwork>