10. Qt – síťové služby

B2B99PPC – Praktické programování v C/C++

Stanislav Vítek

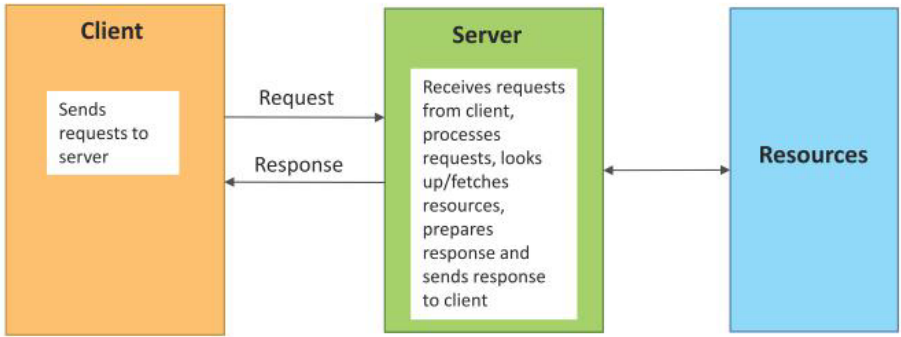
Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Počítačové sítě

- Propojují počítače za účelem výměny informací, používá se specifický protokol
- Jednotlivé vrstvy komunikace popisuje referenční ISO/OSI model
- Třetí (síťová) vrstva – Internet protocol IP
  - Adresování zařízení v síti
  - IPv4 – 32b adresa, 4 dekadická čísla
  - IPv6 – 128b adresa, 8 hexadecimálních čísel
- Čtvrtá vrstva – Transmission control protocol TCP
  - Kontrola doručení, zajištěno správné pořadí dat
  - Nevhodné pro broadcasting
- Čtvrtá vrstva – User datagram protocol UDP
  - Bez kontroly doručení, na rozdíl od TCP neumožňuje opětovné poslání dat
  - Vhodné pro broadcast, multicast, anycast a unicast

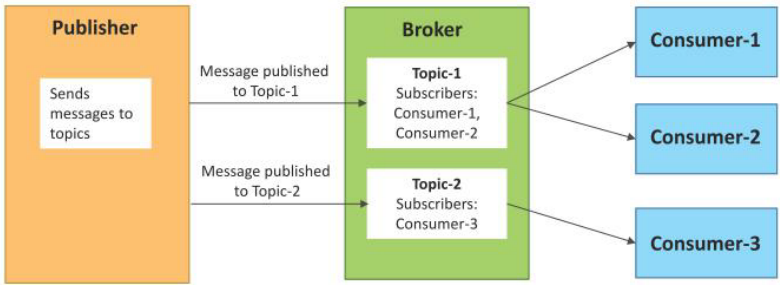
Komunikační model Request-Response

- Request-Response je komunikační model, ve kterém klient posílá požadavky server na tyto požadavky odpovídá.
- Když server obdrží požadavek, rozhodne se, jak odpoví, načte data, načte zdroj reprezentace (např. šablona odpovědi), připraví odpověď a poté odešle odpověď klientovi.



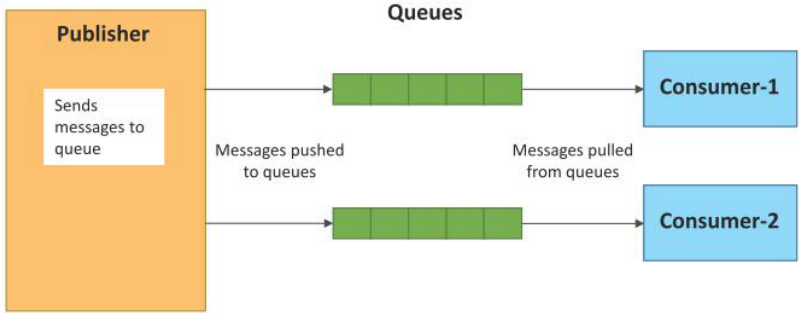
Komunikační model Publish-Subscribe

- Publish-Subscribe je model, který zahrnuje vydavatele (publish), zprostředkovatele (broker) a spotřebitele (consumer) informace.
- Zdrojem dat je vydavatel, který posílá data v rámci téma (topic), které spravuje zprostředkovatel. Vydavatel neví nic o spotřebiteli.
- Spotřebitel se příhlašuje k odběru (subscribe) téma
- Zprostředkoval přeposílá data od vydavatele všem přihlášeným spotřebitelům



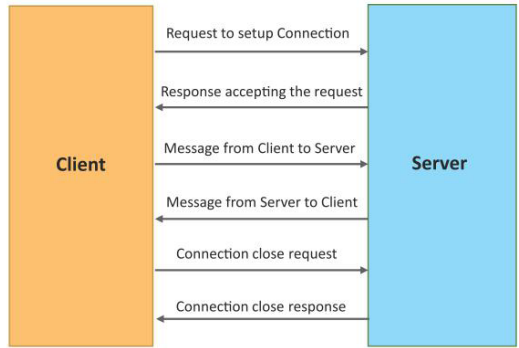
Komunikační model Push-Pull

- Push-Pull je komunikační model, ve kterém producenti dat odesílají data do front a konzumenti je z front odebírají. Producenti nemusí vědět o konzumentech.
- Fronty pomáhají oddělit zasílání zpráv mezi producenty a spotřebiteli.
- Fronty také fungují jako vyrovnávací paměť, která pomáhá v situacích, kdy dochází k výpadkům z důvodu nesouladu mezi rychlostí, kterou producenti odesílají data, a rychlostí, kterou spotřebitelé data stahují.



Komunikační model Exclusive Pair

- Exclusive Pair je obousměrný, plně duplexní komunikační model, který používá trvalé spojení mezi klientem a serverem.
- Jakmile je spojení nastaveno zůstane otevřené, dokud klient odešle požadavek na ukončení.
- Klient a server mohou posílat si navzájem posílat zprávy po navázání spojení.



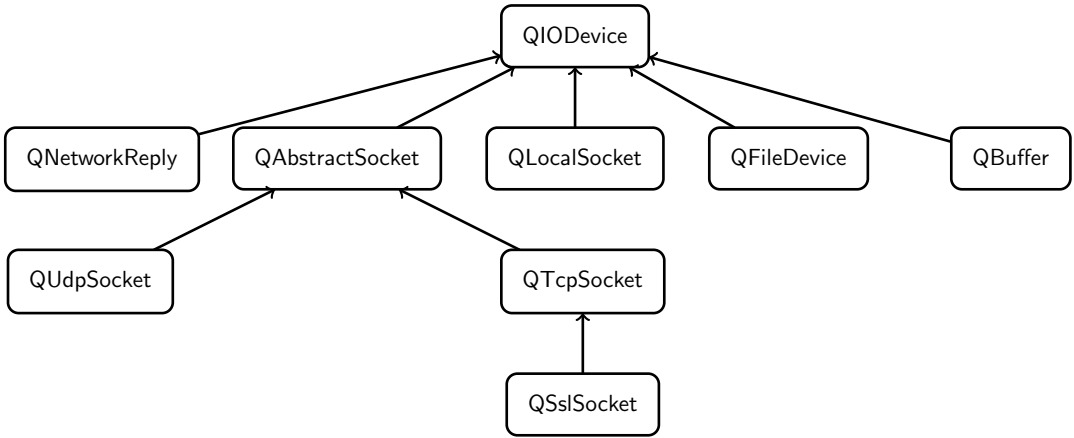
Programování síťových aplikací

- Přístup k síti (networking) je služba poskytovaná OS API (WIN32, POSIX)
- Kód síťových aplikací může značně záviset na platformě
- Privilegované porty
- Komunikace není deterministická – nelze říct kdy a jak bude zpráva doručena
- Řada událostí se děje současně – nutné paralelní zpracování

Síťové aplikace v Qt

- Multiplatformní kód a nástroje, modul `network` (`QT += network`)
- Klíčové třídy:
  - QTcpSocket – zpracuje jedno TCP spojení
  - QTcpServer – vytvoří socket a zpracuje příchozí TCP spojení
  - QUdpSocket – zpracuje UDP spojení (server není třeba)
  - QSslSocket – zpracuje šifrované spojení

Hierarchie tříd



Další třídy: QProcess , ...

Základ komunikace – QAbstractSocket – metody

```
1  bool bind(
2    const QHostAddress &address, quint16 port = 0,
3    BindMode mode = DefaultForPlatform);
4    // mode:  ShareAddress ReuseAddressHint DontShareAddress
5  virtual void connectToHost(
6    const QHostAddress &address, quint16 port,
7    OpenMode mode = ReadWrite,
8    NetworkLayerProtocol protocol = AnyIPProtocol);
9    // mode:  NotOpen ReadOnly WriteOnly Append Truncate Text
10   // protocol:  IPv4Protocol IPv6Protocol
11 virtual void disconnectFromHost();
12 //
13 virtual bool waitForReadyRead(int ms = 30000);
14 //
15 virtual bool waitForBytesWritten(int ms = 30000);
```

Základ komunikace – QAbstractSocket – signály

```
1  void connected();
2  //
3  void disconnected();
4  //
5  void error(QAbstractSocket::SocketError nSocketError);
6  //
7  void hostFound();
8  //
9  void stateChanged(QAbstractSocket::SocketState nSocketState);
```

UDP komunikace – QUdpSocket – čtení a zápis

```
bool hasPendingDatagrams() const;
//
qint64 pendingDatagramSize() const;
//
qint64 readDatagram(
  char *data, qint64 max_size,
  QHostAddress *address = Q_NULLPTR, quint16 *port = Q_NULLPTR);
//
qint64 writeDatagram(
  const char *data, qint64 size,
  const QHostAddress &address, quint16 port);
//
qint64 writeDatagram(
  const QByteArray &datagram,
  const QHostAddress &host, quint16 port);
```

UDP komunikace – sender

```
1  #include <QCoreApplication>
2  #include <QUdpSocket>
3
4  int main(int argc, char ** argv)
5  {
6    QCoreApplication a(argc, argv);
8    QHostAddress address("127.0.0.1"); // QHostAddress::LocalHost
9    QByteArray datagram = "Hello from Sender";
10   QUdpSocket socket;
11   socket.writeDatagram(
12     datagram.data(), datagram.size(), address, 1234);
14   return a.exec();
15 }
```

lec12/01-udp-sender

UDP komunikace – receiver

```
1  QUdpSocket socket;
2  socket.bind(1234, QUdpSocket::ShareAddress);
3  //
4  while (socket.hasPendingDatagrams())
5  {
6    QByteArray datagram;
7    datagram.resize(socket.pendingDatagramSize());
8    QHostAddress address;
9    quint16 port;
10   socket.readDatagram(
11     datagram.data(), datagram.size(), &address, &port);
12   qDebug() << datagram.data();
13 }
```

lec12/01-udp-receiver

UDP komunikace – QUdpSocket – signály

- `readyRead` – na socket dorazila data a lze je zpracovat
- `disconnected` – host se odpojil od socketu

```
1  QObject::connect(&socket, QUdpSocket::readyRead, [&socket](){
2    QByteArray datagram;
3    datagram.resize(socket.pendingDatagramSize());
4    QHostAddress address;
5    quint16 port;
6    socket.readDatagram(
7    datagram.data(), datagram.size(), &address, &port);
8    qDebug() << datagram.data();
9  });
```

TCP komunikace – QTcpServer

- Metody

```
bool listen(
  const QHostAddress &address = QHostAddress::Any,
  quint16 port = 0);
//
virtual QTcpSocket *nextPendingConnection();
//
bool waitForNewConnection(int ms = 0, bool *timeout = Q_NULLPTR);
//
void pauseAccepting();
//
void resumeAccepting();
```

- Signál

```
void newConnection();
```

HTTP protokol

- zkratka z Hyper-Text Transfer Protocol
- možnost přenášet jakákoliv data (soubor, obrázek, výsledek dotazu)
- obvykle provozován nad TCP, port 80 (alternativně 8080)
- existují 4 verze – HTTP/0.9, HTTP/1.0, **HTTP/1.1**, HTTP/2.0

Transakce

```
GET /questions HTTP/1.1

HTTP/1.1 302 Found
Date: Tue, 26 Apr 2022 06:50:44 GMT
Server: Apache/2.4.25 (Debian)
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

HTTP metody

- Protokol HTTP definuje několik metod, pomocí kterých lze na straně serveru požadovat různé typy služeb

OPTIONS	dovoluje klientovi získat informace o možnostech souvisejících se získáním daného prostředku, a schopnostech serveru
GET	získá veškeré informace o zdroji (obvykle stažení HTML obsahu)
HEAD	je obdobou `GET`, ale stahuje se jen hlavička
POST	umožňuje předat data z formuláře
PUT	má obdobnou funkci jako `POST`
DELETE	umožňuje smazat (skrýt) zdroj, pokud to server podporuje
TRACE	možňuje klientovi vidět požadavek přesně tak, jak dorazil na server
CONNECT	se používá pro připojení s proxy servery, které se umí dynamicky přepnout na funkci tunelování (např. SSL tunelování)

Cookies

- Slouží k uchování určitých dat na straně klienta i po zániku spojení
- Klient pošle pomocí metody `POST` serveru nějaká data
- Server je zpracuje, a pošle klientu odezvu obsahující v hlavičce `Set-Cookie`
- Součástí každého dalšího požadavku souvisejícího s daným prostředkem je hlavička s Cookie s údaji zaslanými serverem
- Po ukončení spojení a opětovném navázání obsahuje Cookie data z předchozího spojení, což umožňuje serveru navázat na minulou relaci

Webové služby

- Metoda komunikace dvou elektronických zařízení pomocí webu a vhodného protokolu
- Zpravidla postavena na architektuře server/klient (tzv. WebService)
- Historie
  - 1998
    - WDDX - Web Distributed Data Exchange, založeno na XML
    - XML-RPC
  - 1999
    - SOAP - Simple Object Access Protocol
  - 2000
    - REST - Representational state transfer
  - 2005(1.0), 2009 (2.0)
    - JSON-RPC

SOAP

- HTTP & XML
- WSDL – Web Services Description Language
  - popis funkcí, které nabízí webová služba
  - založeno na XML
- Výhody
  - popíšu webovou službu pomocí WSDL
  - generátor kódu vygeneruje potřebné C/C++ rozhraní, které pak implementuji
- Nevýhody
  - velmi ukecaný protokol
  - fakt hodně
  - pomalé, obtížný (spíše nemožný) přenos binárních dat

JSON-RPC

- JavaScript Object Notation
  - Object: {}
  - Array: []
  - Primitivní datové typy: number, string, bool (true, false), null
- Jednoduché parsování, human readable

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

REST

- REpresentional State Transfer
- Umožňuje přistupovat k datům na určitém místě (zdroj) pomocí standardních metod HTTP (CRUD):
  - Create – vytvoření dat `POST`
  - Retrieve – získání dat `GET`
  - Update – změna dat `PUT`
  - Delete – smazání dat `DELETE`
- Typ dat – v podstatě libovolný
  - binární data
  - XML, JSON, HTML

QNetworkAccessManager

- Metody

```
1  void connectToHost(const QString &hostname, quint16 port = 80);
2  //
3  QNetworkReply *head(const QNetworkRequest &request);
4  //
5  QNetworkReply *get(const QNetworkRequest &request);
6  //
7  QNetworkReply *put(
8    const QNetworkRequest &request, const QByteArray &data);
9  //
10 QNetworkReply *post(
11   const QNetworkRequest &request, const QByteArray &data);
```

- Signál

```
1  void finished(QNetworkReply *pReply);
```

QNetworkRequest/QNetworkReply

- QNetworkRequest

```
1  QVariant header(QNetworkRequest::KnownHeaders header) const;
2  //
3  void setHeader(KnownHeaders header, const QVariant &value);
```

- QNetworkReply

```
1  QVariant header(QNetworkRequest::KnownHeaders header) const;
```

QNetworkRequest::KnownHeaders

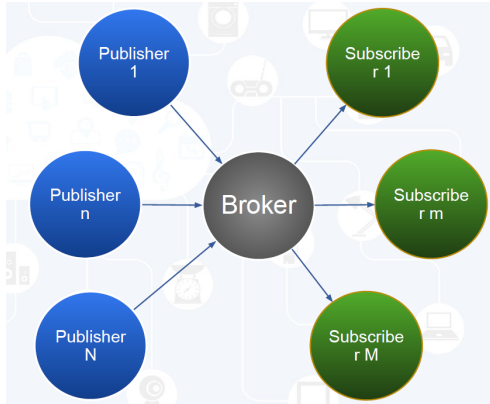
UserAgentHeader	ServerHeader	LastModifiedHeader
CookieHeader	SetCookieHeader	LocationHeader
ContentTypeHeader	ContentDispositionHeader	ContentLengthHeader

MQTT (Message Queue Telemetry Transport Protocol)

- Protokol aplikační vrstvy
  - ve skutečnosti úroveň ISO/OSI 5-7: aplikační, prezentační a relační vrstva
- Je jednoduchý a lehký
- Navržen pro zařízení s omezenou šířkou pásma, nespolehlivou konektivitou a sítě s vysokou latencí
- Používá model Public-subscribe
- Zajišťuje spolehlivost
- Poskytuje některé mechanismy pro zajištění doručení
- Je určen pro sítě TCP/IP (používá TCP)
- Pro sítě bez TCP existuje implementace MQTT-SN

Architektura MQTT

- Vydavatelé a odběratelé vystupují vůči zprostředkovateli jako klienti.
- Zprostředkovatel je mimo firewally
- Zprostředkovatel je centrálním bodem a je kritickou součástí sítě
- Vydavatelé, zprostředkovatel a odběratelé bývají oddělená zařízení a/nebo software
- Vydavatel může současně působit jako odběratel.



MQTT broker

- Broker je vlastně server pro obě části (vydavatele a odběratele).
- Zajišťuje QoS
- Může uchovávat zprávy (data)
- Vydavatel rozhoduje o tom, zda si broker zprávu ponechá.
- V tomto případě každý odběratel při odběru automaticky obdrží nejnovější hodnotu, takže Broker udržuje jakýsi "stav".
- Předává obsah zájemcům z řad odběratelů (obsah pochází od vydavatelů)

MQTT data

- MQTT je textový protokol a je datově agnostický.
- Zprávy (obsah) jsou uspořádány do témat ve formě stromové struktury (jako je adresářová cesta)
  - Oddělovačem je / (lomítko)
- Odběratel se může přihlásit k odběru konkrétního tématu nebo může použít zástupný vzor pro odběr různých témat:
  - `#` znamená celou větev
  - `+` znamená jednoúrovňové
- Příklad
  - Vydavatel vydává např: `CVUT/FEL/209/Sensor/Temperature`
  - Odběratel se přihlásí k odběru: `CVUT/FEL/+/Senzor/#`
  - Odběratel pak bude upozorněn vždy, když zařízení odešle informaci o jakémkoli měření (tj. teploty, ale také vlhkosti a znečištění vzduchu) provedeném `/Senzor/` někde v budově CVUT/FEL (může to být místnost, ale také třeba chodba)

MQTT QoS

- Neuznaná služba
  - Doručeno každému odběrateli maximálně jednou
- Uznávaná služba
  - Zajišťuje doručení zprávy alespoň jednou.
  - Broker očekává potvrzení, jinak zprávu znovu odešle
- Zajištěná služba
  - Dvoustupňové doručení
  - Zajišťuje, že zpráva je každému účastníkovi doručena přesně jednou

Vlastnosti MQTT

- Příznak čisté relace (volitelný) - trvanlivá připojení:
  - Pokud je true, Broker odstraní všechny klientské odběry při odpojení klienta.
  - Pokud je false, spojení zůstane nečinné a všechny zprávy se shromažďují (QoS v závislosti na typu připojení) a doručeny, jakmile je připojení obnoveno.
- Klient může brokerovi nařídit, aby ho nechal odeslat konkrétní téma (nebo témata), když se objeví neočekávané spojení.
  - Zjištění selhání/havárie: vhodné např. pro kritické a bezpečnostní systémy
- Bezpečnost
  - Slabá – uživatelská jména a hesla zasílána v prostém textu
  - Lze využít zabezpečený kanál (SSL/TLS)

MQTT vs HTTP

	MQTT	**HTTP**
Orientace	data	dokumenty
Model komunikace	publish / subscribe	request / response
Komplexita	nízká	vyšší
Velikost zprávy	velmi malá, 2B hlavička	velká
QoS	tři úrovně	všechny zprávy mají stejnou QoS
Distribuce	1-0, 1-1, 1-N	1-1