Komunikační sokety

teorie a implementace v C#, C++ a Javě

Aleš Keprt

Katedra informatiky UP

duben 2006, revize květen 2007

Hrajeme proti sobě…

- …ale jak na to?
- Komunikace mezi procesy na jednom počítači
  - Roury
  - Posílání zpráv oknům ve Windows
  - Sokety
- Komunikace mezi vzdálenými počítači
  - Sokety
  - Komponenty (.NET, COM, CORBA, apod.)

BSD Socket API

- Slogan: „Léty prověřená kvalita“
- Sada funkcí pro komunikaci po počítačové síti
- Používá se nejčastěji s protokoly nad IP
- Výhody:
  - Podpora ve všech operačních systémech
  - Podpora ve všech programovacích jazycích
- Nevýhody:
  - Základní knihovna má jen blokující operace
  - Systém je neobjektový (rok 1983)

Nejprve o těch jazycích…

- Socket API je napsáno pro jazyk C
- BSD Unix má sokety od roku 1983
- Původně komerční, od roku 1989 je BSD volně
- Windows má sokety od roku 1991 (Win 3.11)
- Různé „implementace“ mimo BSD se od té původní mohou lišit, ale zachovávají princip, na kterém to celé funguje
- Moderní jazyky mají implementaci objektovou

A jak je to tedy s těmi jazyky?

- Unix: Socket API je napsáno pro jazyk C
  - C++ na Unixu používá totéž co C
- Windows: obsahují Winsock knihovnu
  - Opět primárně pro C a C++, ostatní jazyky to mohou převzít (např. Delphi)
- C#/Visual Basic/J# používají .NET
  - .NET obsahuje objektovou verzi soketů
- Java má sokety zabudované ve své knihovně
  - Opět jde o objektovou implementaci soketů
- Common Lisp nemá sokety ve standardu
  - Mnohé implementace (jako Lispworks) sokety mají doplněny nad rámec standardu

Trocha teorie

- Sokety obvykle používáme nad protokolem IP
- Komunikujeme spojeně nebo rozpojeně
- Spojeně: protokol TCP
  - Na začátku navážeme spojení
  - Data pak posíláme stejně, jako když se pracuje se soubory
  - Důsledek: TCP zaručuje doručení každé zprávy, včetně správného pořadí zpráv
- Rozpojeně: protokol UDP
  - Posíláme samostatné balíčky dat
  - U každého balíčku uvedeme adresáta
  - UDP nezaručuje doručení, ani pořadí zpráv

Adresování vzdálených počítačů

- „Kdo by tohle neznal?“ ☺
- Každý počítač má IP adresu (4 čísla, tj. 32 bitů)
- Spojení navazujeme vždy na nějaký port
  - Port je číslo identifikující spojení
  - Každé spojení je mezi dvěma síťovými uzly
  - Každý z nich má svou IP adresu a svůj port
  - Většinou zadáme jen adresu a port vzdáleného počítače, lokální jsou doplněny automaticky
    - Naše IP adresa je většinou jasná ☺
    - Náš port nezadáváme, protože je nám to jedno ☺
- Pozor! Ne-IP protokoly to mohou mít jinak

Co je to soket?

- Soket je jeden konec komunikačního kanálu
- Každé aktivní spojení má dva sokety
  - Nespojovaná komunikace taktéž dva
- Každý soket má lokální adresu, port a protokol
  - Je-li připojený, má i vzdálenou adresu a port
- Zajímají nás hlavně TCP sokety
  - Jsou dva druhy TCP soketů
    - Naslouchací soket pro příjem žádostí o spojení
    - Klientský soket pro běžnou komunikaci

Algoritmus spojování (TCP)

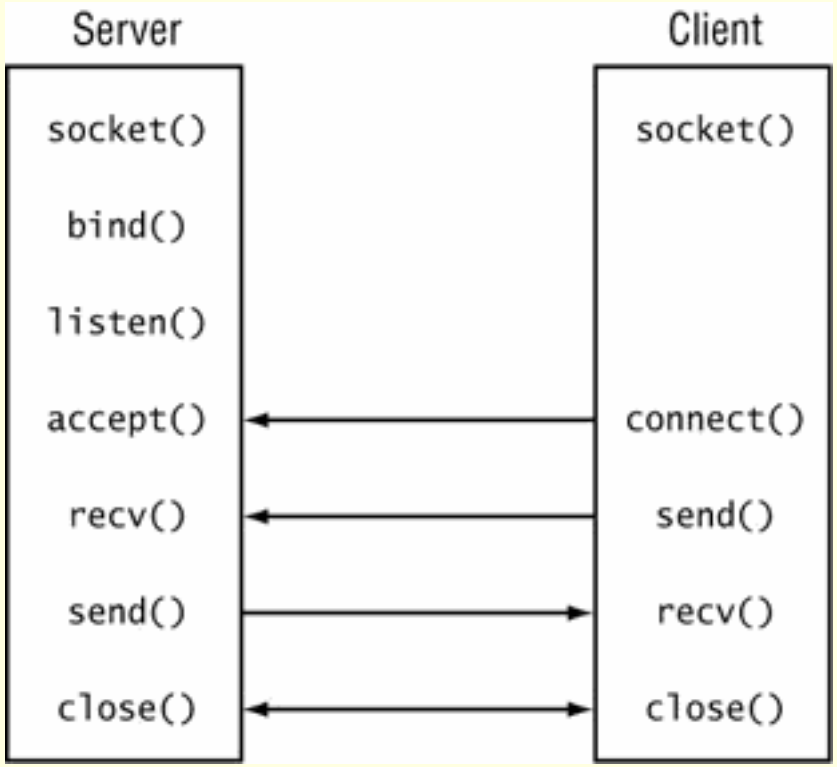
- Jeden počítač je HOST (hostitel) neboli server
- Libovolné další počítače jsou CLIENT (klient)

1. Host otevře naslouchací soket a čeká spojení
   - Zadá lokální port, na kterém očekává spojení
2. Klient otevře běžný soket a připojí se k hostu
   - Zadá vzdálenou adresu a port, kam se připojit
3. Připojením dostane host adresu a port klienta
   - Tím mu také vznikne běžný soket
   - Běžné sokety jsou nyní propojeny
4. Host může nyní čekat na připojení dalšího klienta
5. Komunikace končí, když jedna strana zavře soket

Jak probíhá komunikace

- Komunikační kanál je obousměrný
- Každá strana může libovolně zapisovat i číst
- Když pošleme víc zpráv krátce po sobě, na klienta dojde jedna spojená zpráva
- Funkce pro čtení i zápis jsou blokující
  - Funkce se vrací, až jsou data poslána/přijata
  - Někdy to může být dost problém – budeme to řešit později ☺

Příkazy BSD Socket API



- socket() založí socket nějakého protokolu
- bind() sváže soket s lokální adresou a portem
- listen() nastaví soket do naslouchacího režimu
- accept() přijme žádost o spojení, vrací soket
- recv/send komunikují
- connect() připojí k hostiteli

Hrajeme dámu po síti

- Musíme si rozmyslet náš „protokol“
- Protokol = pravidla chování
- Zkusíme to takhle:
  1. Posíláme textové zprávy (kódování 8bit)
  2. První bajt ve zprávě je délka zbytku zprávy
  3. Klient vždy čeká na příkaz od hosta
  4. Po provedení příkazu klient odešle odpověď
  5. Kdokoliv může kdykoliv poslat „quit“, čímž ukončí spojení

Příkazy našeho protokolu

- Příkazy dark a light nastaví barvu, za kterou bude hráč hrát
  - Odpověď: ok
- Příkaz yourmove se dotáže na tah
  - Odpověď má tvar mymove a1b2
- Příkaz move a1b2 oznámí tah druhého hráče
  - Odpověď: ok

- Nyní si to můžeme předvést v praxi ☺

Klient v C#

- .NET má třídu Socket reprezentující soket a třídu TcpClient pro snazší práci s TCP protokolem
- Nejprve připojíme

  TcpClient socket = new TcpClient(host, port);
- Můžeme posílat zprávy

  socket.Client.Send(byte[] buffer);
- Můžeme přijímat zprávy

  socket.Client.Receive(byte[] buffer);
- Zavřeme soket

  socket.Close();

vrací počet bajtů

Klient v Javě

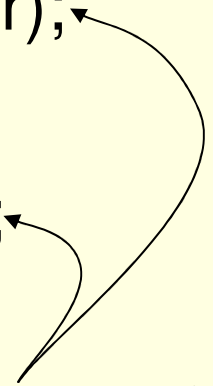
- Java má třídu Socket reprezentující soket
- Je to velmi podobné .NETu ☺
- Nejprve připojíme

  Socket socket = new Socket(host, port);
- Můžeme posílat zprávy

  socket.getOutputStream().write(char[] buffer);
- Můžeme přijímat zprávy

  socket.getInputStream().read(char[] buffer);
- Zavřeme soket

  socket.close();



Klient v C/C++

- Co nám C# a Java udělá pomocí čtyř příkazů, na to v C/C++ potřebujeme 50 řádků kódu ☹
- První problém: názvy síťových adres
  - Je nutno složitě rozlišovat mezi číslem a jménem
- Druhý problém: není to objektové
  - Nemáme žádnou entitu, která by si kompletně pamatovala stav soketu
  - Musíme pracovat zvlášť se soketem a adresami
  - Místo výjimek to jen vrací různá chybová čísla ☹

Klient v C/C++ – adresa hostitele

- Nejprve sestavíme adresu hostitele

```
HOSTENT *host = 0;
unsigned long ip = inet_addr(hostname);
if(ip!=INADDR_NONE) {
    //číselná IP adresa
    host = gethostbyaddr((char*)&ip, 4, AF_INET);
} else {
    //jmenná adresa
    host = gethostbyname(hostname);
}
if(host==0) throw NetException("Can't find host");
```

Klient v C/C++ – vytvoření soketu

```
SOCKET client_socket = socket(AF_INET, SOCK_STREAM,
   IPPROTO_TCP);
if(client_socket == INVALID_SOCKET) { … }

SOCKADDR_IN server_sin;
server_sin.sin_family = AF_INET;
server_sin.sin_addr = *((LPIN_ADDR)*host->h_addr_list);
server_sin.sin_port = htons(port); //port number

if(SOCKET_ERROR == connect(client_socket,
   (SOCKADDR*)&server_sin, sizeof(SOCKADDR))) { … }
```

ošetření chyby

Klient v C/C++ – práce se zprávami

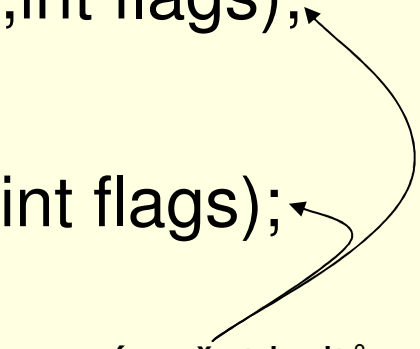
- Posíláme zprávy

  send(SOCKET s,const char*buf,int len,int flags);
- Přijímáme zprávy

  recv(SOCKET s,const char*buf,int len,int flags);
- Zavřeme soket

  closesocket(SOCKET s); //Windows

  close(int s); //BSD



Klient v C/C++ – přenositelnost

- Jak známo C/C++ jsou velmi „přenositelné“
  - Proto musíme program pro každý operační systém speciálně upravit ☹
- Windows (Winsock)
  - Na začátku musíme volat WSAStartup(ver,&data);
    - Zde zvolíme požadovanou verzi Winsock
  - Na konci voláme WSACleanup();
  - Všechny datové struktury jsou big endian 
- BSD
  - Místo SOCKET se používá přímo int 
    (jazyk C je stejně jen slabě typovaný)

Server v C#

- Nejprve vytvoříme naslouchací soket

  TcpListener listener = new TcpListener(port);
- Začneme naslouchat

  listener.Start();
- Přijmeme spojení

  TcpClient client = listener.AcceptTcpClient();
  - Objekt client je běžný TCP soket
- Ukončíme naslouchací soket

  listener.Close();
- Poznámka: V C/C++ to určitě dělat nebudeme ☺

Neblokující sokety

- Výchozí režim soketů je blokující
- Blokující operace „zasekne“ program
  - Týká se operací accept, recv, send
- Neblokující operace můžeme udělat několika způsoby
  1. Neblokující sokety – pomocí ioctl(), toto řešení je systémově závislé
  2. Multiplexování soketů – styl „wait for many“
  3. Pomocí událostí – jen ve Windows
  4. Pomocí vláken – vytvoříme víc vláken, takže pak nevadí, když se vlákno „zasekne“

Další čtení

- Richard Blum: *C# Network Programming.* Sybex, 2002, ISBN 0782141765.
- M. Tim Jones: *BSD Sockets Programming from a Multi-Language Perspective.* Charles River Media, 2003, ISBN 1584502681.