

# Programování síťové služby

## HTTP nástěnka

# Obsah

Cíl projektu.....	2
Popis problematiky.....	2
Architektura Klient-Server .....	2
Aplikační vrstva .....	2
HTTP .....	3
Historie .....	3
Verze 1.1 .....	3
Základní dotazovací metody.....	3
Přehled kódů odpovědí od serveru .....	3
API .....	4
Definice .....	4
Popis API naší aplikace.....	4
Implementace .....	5
Použité knihovny .....	5
Server .....	5
Klient.....	5
Použití.....	6
Server .....	6
Klient.....	6
Testování .....	7
Manuální testování .....	7
Postman .....	7
Závěr.....	7
Struktura projektu.....	8
Reference.....	10

# Cíl projektu

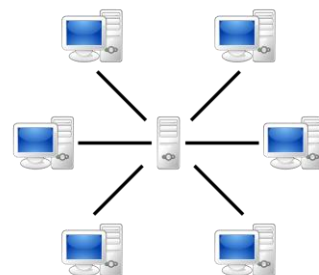
Cílem tohoto projektu bylo vytvořit server na kterém bude možné ukládat nástěnky, potažmo příspěvky na těchto nástěnkách. Uživatelé prostřednictvím klientské aplikace budou moci provádět operace nad těmito nástěnkami a jejich příspěvky. Operacemi se rozumí vytvoření nové nástěnky, smazání existující nástěnky, výpis seznamu všech existujících nástěnek, vložení nového příspěvku na některou z nástěnek, smazání některého příspěvku z nástěnky a možnost editace obsahu příspěvku na nástěnce.

## Popis problematiky

### Architektura Klient-Server

Architektura Klient-Server je síťová architektura, která odděluje poskytovatele zdroje nebo služby takzvaný server a žadatele o jeho služby takzvaného klienta. Klient se serverem obvykle komunikuje pomocí počítačové sítě, ale není to podmínkou. Komunikace mezi serverem a klientem funguje na základě zasílání zpráv, přičemž zprávy které jsou odeslané klientem nazýváme požadavky, na které server následně odpoví zprávou kterou nazýváme odpověď.

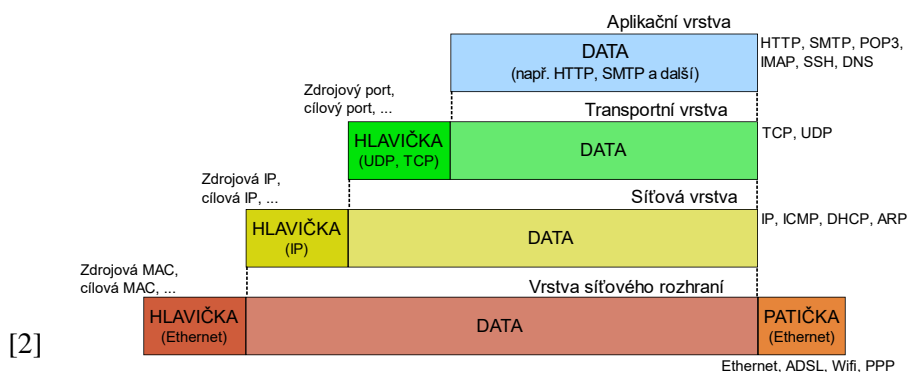
[1]



### Aplikační vrstva

Aplikační vrstva je nejvyšší vrstvou TPC/IP respektive ISO/OSI modelu. Aplikační protokoly vždy používají jednu ze základních služeb transportní vrstvy tz. TCP nebo UDP (případně kombinace obou DNS). Pro rozlišení aplikačních protokolů se používají tzv. porty, což jsou číselná označení aplikací. Každopádně tedy síťové spojení aplikace lze jednoznačně určit pomocí transportního protokolu, adresy počítače a onoho čísla portu. Typickými představiteli protokolů na aplikační vrstvě jsou například HTTP, SSH, FTP, SMTP apod..

### ZAPOUZDŘENÍ DAT V SÍTI TCP/IP



[2]

# HTTP

HTTP (z anglického Hypertext Transfer Protocol) je protokol na aplikační vrstvě. Tento protokol funguje na principu dotaz-odpověď, přičemž tento protokol je základem pro WWW komunikaci. Jedná se o bezstavový protokol.<sup>1</sup> Slouží především k přenosu hypertextových dokumentů (formát HTML), souborů formátu XML, JSON apod. Používá obvykle port TCP/80, verze HTTP 1.1.

## Historie

Historie HTTP protokolu sahá do roku 1991. Zpětně byla verze popsána v tomto roce označena za verzi 0.9. Tato verze obsahovala pouze jednu metodu, a to metody `GET` s pouze jedním parametrem kterým byl název požadovaného dokumentu. Odpověď serveru byla bez hlaviček HTTP, vracel se pouze požadovaný dokument.

## Verze 1.1

Tato verze je použita pro řešení našeho projektu. Popis této verze HTTP lze nalézt v [RFC2616](https://tools.ietf.org/html/rfc2616). V této verzi lze na rozdíl od verzí předchozích provozovat více WWW serverů na jedné adrese (odlišení pomocí host), dále zde byly přidány dotazovací metody `DELETE`, `OPTION`, `CONNECT` a `TRACE`. Posledními podstatnými změnami je podpora přednášení více souborů na jedno spojení plus podpora keep-alive connection<sup>2</sup>. [3]

## Základní dotazovací metody

- **GET** – Požadavek klienta na uvedený objekt serveru. Jedná se obecně o nejpoužívanější dotazovací metodu v protokolu HTTP, tato metoda je výchozí při zobrazení WWW stránek.
- **POST** – Odesílá data na server, typické použití je například odeslání dat z formuláře. S předávaným objektem se zachází obdobně jako při použití metody `GET` (i tato může též odesílat data na server). Pomocí metody `POST` je možné odesílat data větší než 512B, dále je tato metoda vhodná pro posílání dat které není vhodné přenášet jako součást URL (případ metody `GET`).
- **DELETE** – Smazání uvedeného objektu ze serveru (nutnost jistých oprávnění).
- **PUT** – Nahraje data na server (opět je zde nutnost jistých oprávnění), zřídka používané běžně se pro nahrávání dat používá FTP.<sup>3</sup>

## Přehled kódů odpovědí od serveru

- **100 – 199** Informační odpovědi
- **200 – 299** Odpovědi značící úspěch
- **300 – 399** Přesměrování
- **400 – 499** Chyba způsobená na straně klienta
- **500 – 599** Chyba způsobená na straně serveru

Bližší popis lze nalézt na této adrese [RFC2616](https://tools.ietf.org/html/rfc2616).

---

<sup>1</sup> Bezstavovým protokolem rozumíme takový protokol, kde význam jednotlivých dotazů na server mezi sebou nemá souvislost, protokol tyto informace neobsahuje.

<sup>2</sup> Zachování aktivního TCP spojení <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Keep-Alive>

<sup>3</sup> File Transfer Protocol <https://tools.ietf.org/html/rfc959>

# API

## Definice

API (z anglického Application programming interface), je označení pro rozhraní, nebo komunikační protokol mezi klientem a serverem. API si můžeme představit jako “smlouvu” mezi serverem a klientem, pokud klient odešle specifický požadavek ve správném formátu, dostane odpověď ve specifickém formátu, nebo se zahájí předem definovaná akce. Cílem takového API je zjednodušení tvorby klientské části software [4].

## Popis API naší aplikace

V následujícím popisu se budou objevovat následující zástupné reprezentace.

**<name>** - Je jménem nástěnky.

**<id>** - Je aktuální pořadí konkrétního příspěvku v nástěnce **<name>**.

**<content>** - Je obsah (hodnota) příspěvku.

- **'GET /boards'** - Vrátí seznam dostupných nástěnek (jedna nástěnka/řádek).
- **'POST /boards/name'** - Vytvoří novou nástěnku s názvem name.
- **'DELETE /boards/name'** - Smaže nástěnku se jménem name a všechn její obsah.
- **'GET /board/name'** - Vrátí všechny příspěvky na nástěnce se jménem name.
- **'POST /board/name'** + 'obsah příspěvku v těle HTTP zprávy' - Přidá příspěvek na nástěnku se jménem name.
- **'PUT /board/name/id'** - Smaže z nástěnky se jménem name příspěvek s aktuálním pořadovým číslem id.
- **'DELETE /board/name/id'** + 'obsah příspěvku v těle HTTP zprávy' - Aktualizuje obsah příspěvku s aktuálním pořadovým číslem id na nástěnce se jménem name.

Server následně odpoví hlavičkou s návratovým kódem, případě i obsahem.

- **200** - Úspěch metod GET, PUT a DELETE.
- **201** - Úspěch metody POST.
- **404** - Nástěnka se jménem name, nebo příspěvek s s pořadovým číslem id v nástěnce name neexistuje.
- **400** - POST nebo PUT metoda s délkou příspěvku 0.
- **409** - Pokus o vytvoření nástěnky se jménem name, avšak nástěnka se stejným jménem již existuje.
- **404** - Neznámý požadavek.

# Implementace

Při implementaci byly použity informace, respektive příklady, které jsou dostupné na stránkách předmětu Síťové aplikace s správou sítí v sekci Příklady. V obou částech projektu byl pro funkci `recv()` použit buffer o statické velikosti 65536B. Tato velikost byla odvozena z diskuze na programátorském fóru StackOverflow a dle mého názoru je dostatečná pro použití v aplikaci tohoto typu při očekávání normálního užití. [5]

## Použité knihovny

```
+ <mutex>
+ <cerrno>
+ <iostream>
+ <string>
+ <regex>
+ <thread>
+ <netdb.h>
+ <sys/socket.h>
+ <arpa/inet.h>
+ <netinet/in.h>
```

## Server

Server je tedy implementován tak, že po jeho spuštění se jako první provede kontrola vstupních parametrů tohoto programu. Jediné dvě přípustné možnosti spuštění serveru jsou buďto s přepínačem `-h` (pouze vypsání uživatelské nápovědy), nebo s přepínačem `-p` následovaným číslem portu na kterém má server očekávat příchozí spojení. Po spuštění serveru s přepínačem `-p` dojde tedy ke kontrole čísla portu, zda se jedná o validní číslo portu. Pokud je toto číslo validní proběhne snaha o započetí naslouchání na daném portu. Pokud na server dojde požadavek, vytvoří se nové vlákno, ve kterém bude tento příspěvek zpracováván. V tomto vlákne proběhne následně kontrola, zda se jedná o nějaký známý požadavek pro server (viz API), pokud ano tak proběhnou další kontroly validity dotazu (specifické pro daný dotaz) a následně server odpoví klientovi. Jedná se tedy o konkurentní implementaci serveru. Samotné nástěnky a jejich příspěvky jsou na serveru uloženy ve dvourozměrném vektoru. Jak již bylo zmíněno, jelikož se jedná o konkurentní implementaci, je třeba zajistit konzistenci dat, při přístupu více klientů současně k tomuto sdílenému zdroj. Za tímto účelem jsem se rozhodl použít mutexy (zámky) [6].

## Klient

Po spuštění klientské části aplikace, opět jako první dojde k ověření vstupních parametrů. Kontrola některých vstupních parametrů by nemusela probíhat na straně klienta, ale až na straně serveru. Rozhodl jsem se ovšem některé tyto kontroly implementovat již na klientské straně aplikace, za účelem snížení množství nevalidních dotazů na server a tím jeho zatěžování. Po kontrole vstupních parametrů tedy dojde k vytvoření HTTP dotazu podle parametrů zadaných uživatelem. Následně proběhne snaha a započetí komunikace se serverem na zadaném portu a odeslání požadavku na tento server. Pokud server klientské části aplikace odpoví, je hlavička této odpovědi vypsána na standardní chybový výstup a samotné tělo těla HTTP odpovědi je vypsáno na standardní výstup.

# Použití

Nejprve je nutné projekt zkompileovat pomocí příkazu `make`, který zavolá přiložený Makefile. Další doplňující informace jsou obsaženy v souboru README.

## Server

**Spuštění:** `./isaserver -p <port_number>`

**<port>** je číslo portu na kterém má server očekávat spojení (**<port>** musí být v rozsahu <0;65535>)

**Příklad spuštění:**

```
./isaserver -p 12345
```

## Klient

(Pro funkčnost klienta musí být zapnutý server)

Hlavička HTTP odpovědi od serveru je vypsána na **stderr**, tělo HTTP zprávy na **stdout**

**Spuštění:** `./isaclient -H <host> -p <port> <command>`

**<host>** je hostname / IP adresa verze 4 serveru

**<port>** je číslo portu na kterém server očekává spojení (**<port>** musí být v rozsahu <0;65535>)

**<command>** může být jedna z následujících variant

1) **boards** - Vrátí uživateli seznam všech dostupných nástěnek (1/řádek)

2) **board add <name>** - Vytvoří na serveru novou nástěnku s názvem **<name>**

3) **board delete <name>** - Smaže ze serveru nástěnku s názvem **<name>** a všechnen její obsah

4) **boards list <name>** - Vrátí obsah z dané nástěnky se jménem **<name>** (jeden příspěvek / řádek)

5) **item add <name> <content>** - Přidá na nástěnku s názvem **<name>** příspěvek **<content>**  
(Pokud není zadán content, odešle se prázdný)

6) **item delete <name> <id>** - Odstraní z nástěnky se jménem **<name>** příspěvek s číslem (pořadím) **<id>** (celé kladné číslo > 0)

7) **item update <name> <id>** - Aktualizuje na nástěnce se jménem **<name>** příspěvek s číslem (pořadím) **<id>** (celé kladné číslo > 0)

**Omezení :**

Nezáleží, jestli je nejprve zadán hostname a následně číslo portu nebo naopak. Pokud chceme mezeru nebo znak nového řádku (/n) ve jméně nástěnky nebo v obsahu příspěvku je třeba toho uvést do dvojitéch uvozovek .. " ..."

**Příklad spuštění:**

```
./isaclient -H localhost -p 12345 board add nastenkaA
./isaclient -H localhost -p 12345 item add nastenkaA "\"Toto je
viceslovny prispevek\""
```

# Testování

## Manuální testování

Za účelem otestování správného chování serveru jsem si vytvořil jednoduchý skript v jazyku Python. Pomocí tohoto skriptu jsem na server nahrál několik nástěnek a na každou nástěnku několik příspěvků, abych nemusel po každém znovuspuštění nebo opravě chyby manuálně nahrávat tyto nástěnky na server. Následně jsem se pokoušel nad nástěnkami a jejich příspěvky provádět provádět zadané operaci a kontroval správnost výstupu a kódu v http hlavičce.

Během samotné implementace jsem také pro odhalení chyb v implementaci používal nejruznější kontrolní výpisy, které se vypisovaly při aktivním „DEBUG flagu“. Tyto výpisy byly v konečné verzi aplikace odstraněny. Velice nápomocen mi byl také nástroj Wireshark<sup>4</sup>.



## Postman

Z důvodu, že server by neměl potencionálně očekávat dotazy pouze od naší verze klientské aplikace, rozhodnul jsem se po doporučení kolegy toto simulovat pomocí nástroje Postman, který umožňuje intuitivní zasílání dotazů, tak jejich přehlednou vizualizaci včetně potencionálních odpovědí serveru.<sup>5</sup>



## Závěr

V rámci projektu jsme si tedy vyzkoušeli implementaci jednoduché verze aplikace typu klient server v jazyce C++. Aplikace dle mého otestování splňuje zadané požadavky na funkcionalitu. Pro potencionální praktické využití aplikace, by bylo jistě minimálně nutné vytvořit grafické uživatelské rozhraní pro komfortnější uživatelskou interakci.

---

<sup>4</sup> Wireshark download: <https://www.wireshark.org/>

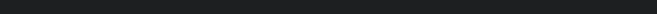
<sup>5</sup> Postman download: <https://www.getpostman.com/>



# Struktura projektu

- + isaclient.cpp
- + isaclient.h
- + isaserver.cpp
- + isaserver.h
- + Makefile
- + README
- + manual.pdf

# Snímky



```
martin@martin-dell: ~/Desktop/Projects/ISA projekt/src
martin@martin-dell:~/Desktop/Projects/ISA projekt/src$ ./isaserver -p 12345
```

```
martin@martin-dell: ~/Desktop/Projects/ISA projekt/src
martin@martin-dell:~/Desktop/Projects/ISA projekt/src$ ./isaclient -p 12345 -H localhost board add ISApjekt2019 2>&1
HTTP/1.1 201 OK

martin@martin-dell:~/Desktop/Projects/ISA projekt/src$ ./isaclient -p 12345 -H localhost item add ISApjekt2019 prispevek1 2>&1
HTTP/1.1 201 OK

martin@martin-dell:~/Desktop/Projects/ISA projekt/src$ ./isaclient -p 12345 -H localhost item add ISApjekt2019 prispevek2 2>&1
HTTP/1.1 201 OK

martin@martin-dell:~/Desktop/Projects/ISA projekt/src$ ./isaclient -p 12345 -H localhost item add ISApjekt2019 prispevek3 2>&1
HTTP/1.1 201 OK

martin@martin-dell:~/Desktop/Projects/ISA projekt/src$ ./isaclient -p 12345 -H localhost boards 2>&1
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 15

ISApjekt2019
martin@martin-dell:~/Desktop/Projects/ISA projekt/src$ ./isaclient -p 12345 -H localhost board list ISApjekt2019 2>&1
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 59

[ISApjekt2019]
1. prispevek1
2. prispevek2
3. prispevek3
martin@martin-dell:~/Desktop/Projects/ISA projekt/src$
```

Capturing from Loopback: lo

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 12345 Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
338	513.660668434	127.0.0.1	127.0.0.1	TCP	66	12345 → 40338 [ACK] Seq=21 Ack=120 Win=65536 Len=0 TSval=4058
348	517.507243413	127.0.0.1	127.0.0.1	TCP	74	40342 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM
349	517.507265684	127.0.0.1	127.0.0.1	TCP	74	12345 → 40342 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549
350	517.507285176	127.0.0.1	127.0.0.1	TCP	66	12342 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4058671
351	517.507387611	127.0.0.1	127.0.0.1	HTTP	184	POST /board/ISaprojekt2019 HTTP/1.1 (text/plain)
352	517.507481516	127.0.0.1	127.0.0.1	TCP	66	40345 → 40342 [ACK] Seq=1 Ack=119 Win=65498 Len=0 TSval=40586
353	517.511392389	127.0.0.1	127.0.0.1	HTTP	85	HTTP/1.1 201 OK
354	517.511419421	127.0.0.1	127.0.0.1	TCP	66	40342 → 12345 [ACK] Seq=119 Ack=20 Win=65536 Len=0 TSval=4058
355	517.511452665	127.0.0.1	127.0.0.1	TCP	66	12345 → 40342 [FIN, ACK] Seq=20 Ack=119 Win=65536 Len=0 TSval
356	517.512159425	127.0.0.1	127.0.0.1	TCP	66	40342 → 12345 [FIN, ACK] Seq=119 Ack=21 Win=65536 Len=0 TSval
357	517.512173783	127.0.0.1	127.0.0.1	TCP	66	12345 → 40342 [ACK] Seq=21 Ack=120 Win=65536 Len=0 TSval=4058
358	521.086130986	127.0.0.1	127.0.0.1	TCP	74	40344 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM
359	521.086148110	127.0.0.1	127.0.0.1	TCP	74	12345 → 40344 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549
360	521.086161998	127.0.0.1	127.0.0.1	TCP	66	40344 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4058674
361	521.086230874	127.0.0.1	127.0.0.1	HTTP	184	POST /board/ISaprojekt2019 HTTP/1.1 (text/plain)
362	521.086239313	127.0.0.1	127.0.0.1	TCP	66	12345 → 40344 [ACK] Seq=1 Ack=119 Win=65498 Len=0 TSval=40586
363	521.088833527	127.0.0.1	127.0.0.1	HTTP	85	HTTP/1.1 201 OK
364	521.088846390	127.0.0.1	127.0.0.1	TCP	66	40344 → 12345 [ACK] Seq=119 Ack=20 Win=65536 Len=0 TSval=4058
365	521.088861129	127.0.0.1	127.0.0.1	TCP	66	12345 → 40344 [FIN, ACK] Seq=20 Ack=119 Win=65536 Len=0 TSval
366	521.089340191	127.0.0.1	127.0.0.1	TCP	66	40344 → 12345 [FIN, ACK] Seq=119 Ack=21 Win=65536 Len=0 TSval
367	521.089353336	127.0.0.1	127.0.0.1	TCP	66	12345 → 40344 [ACK] Seq=21 Ack=120 Win=65536 Len=0 TSval=4058
368	528.070584955	127.0.0.1	127.0.0.1	TCP	74	40346 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM
369	528.070699131	127.0.0.1	127.0.0.1	TCP	74	12345 → 40346 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549
370	528.070632412	127.0.0.1	127.0.0.1	TCP	66	40346 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4058681
371	528.070735670	127.0.0.1	127.0.0.1	HTTP	113	GET /boards HTTP/1.1
372	528.070750209	127.0.0.1	127.0.0.1	TCP	66	12345 → 40346 [ACK] Seq=1 Ack=48 Win=65536 Len=0 TSval=405868
373	528.072591744	127.0.0.1	127.0.0.1	HTTP	146	HTTP/1.1 200 OK (text/plain)
374	528.072599120	127.0.0.1	127.0.0.1	TCP	66	40346 → 12345 [ACK] Seq=48 Ack=81 Win=65536 Len=0 TSval=40586
375	528.072613941	127.0.0.1	127.0.0.1	TCP	66	12345 → 40346 [FIN, ACK] Seq=81 Ack=48 Win=65536 Len=0 TSval=
376	528.072595993	127.0.0.1	127.0.0.1	TCP	66	40346 → 12345 [FIN, ACK] Seq=48 Ack=81 Win=65536 Len=0 TSval=
377	528.072969298	127.0.0.1	127.0.0.1	TCP	66	12345 → 40346 [ACK] Seq=82 Ack=49 Win=65536 Len=0 TSval=40586
378	556.174899046	127.0.0.1	127.0.0.1	TCP	74	40348 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM
381	556.174941870	127.0.0.1	127.0.0.1	TCP	74	12345 → 40348 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549
382	556.174971774	127.0.0.1	127.0.0.1	TCP	66	40348 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval

# Reference

- [1] Mauro Bieg, „Client Server Wikipedia,“ Wikipedia, [Online]. Available: <https://commons.wikimedia.org/w/index.php?curid=2551745>. [Přístup získán 19 10 2019].
- [2] „Schéma zpouzdření aplikačních dat na jednotlivých vrstvách rodiny protokolů TCP/IP,“ [Online]. Available: [https://commons.wikimedia.org/wiki/File:Tcpip\\_zapouzdeni.svg](https://commons.wikimedia.org/wiki/File:Tcpip_zapouzdeni.svg). [Přístup získán 19 10 2019].
- [3] „HTTP Wikipedia,“ Wikipedia, [Online]. Available: [https://cs.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol). [Přístup získán 20 10 2019].
- [4] „API Wikipedia,“ Wikipedia, [Online]. Available: [https://en.wikipedia.org/wiki/Application\\_programming\\_interface](https://en.wikipedia.org/wiki/Application_programming_interface). [Přístup získán 18 10 2019].
- [5] StackOverflow, „How large should my recv buffer be when calling recv in the socket library,“ StackOverflow, [Online]. Available: <https://stackoverflow.com/questions/2862071/how-large-should-my-recv-buffer-be-when-calling-recv-in-the-socket-library>. [Přístup získán 16 10 2019].
- [6] V. Hordějčuk, „http://voho.eu/,“ [Online]. Available: <http://voho.eu/wiki/paralelismus/>. [Přístup získán 19 10 2019].