VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

Síťové aplikace a správa sítí

2019/2020

Projekt

Varianta 3: HTTP Nástěnka

Pavel Chlubna (xchlub02)

Brno, 17. listopadu 2019

Obsah

Úvod

Tento projekt se zabývá vytvořením server a klient aplikace, které spolu budou vzájemně komunikovat pomocí HTTP protokolu a API nad ním. O protokolu HTTP a toto API a způsob, jak se s ním pracuje v klient aplikaci bude popsáno níže.

Server aplikace bude na své straně uchovávat nástěnky s příspěvky a klient aplikace k nim bude moci přistupovat a modifikovat je.

Projekt se dá rozdělit do dvou částí a to vytvoření serverové aplikace a klient aplikace. Serverová část naslouchá na zadaném portu a čeká na požadavek, který přijde od klienta pomocí klient aplikace.

K tomu abych docílil správnému uchovávání dat jsem aplikoval jednosměrně vázaný lineární seznam a k posílání HTTP protokolu přes síť BSD socket. Jak byly tyto technologie použity si ukážeme až později.

Zadání

Aplikace umožňuje klientovi spravovat nástěnky na serveru pomocí HTTP API. API jim umožňuje procházet, přidávat, upravovat a mazat příspěvky na nástěnkách a stejně tak nástěnky samotné. Nástěnka obsahuje seřazený seznam textových příspěvků.

Nástěnkou se rozumí uspořádaný seznam textových (ASCII) příspěvků. Každý příspěvek má **ID** (číslované od 1) a textový obsah, který může být víceřádkový. **ID** není permanentní, koresponduje s aktuální pozicí v seznamuů. Operace nad nástěnkou by neměly měnit pořadí příspěvků. Název nástěnky může obsahovat znaky **a-z**, **A-Z**, a **0-9**.

Aplikace mezi sebou komunikují pomocí protokolu HTTP a rozhraním nad ním. Použitá verze HTTP je **1.1**. Stačí použít minimální počet hlaviček potřebných ke správné komunikaci, aplikace by však měla být schopna vypořádat se také s neznámými hlavičkami (t.j. přeskočit a ignorovat). V případě, kdy se posílají data je použit **Content-Type: text/plain**. Riadky obsahu jsou oddělené pouze **\n**.

HTTP Protokol

Jedná se o protokol aplikační vrstvy, který slouží ke komunikaci po internetu především se servery WWW. Obecně tento protokol používá port TCP/80. Umí přenášet dokumenty ve formátu HTML a XML (i jiných), ale v našem projektu jsem pomocí tohoto protokolu přenášel pouze prostý text.
Samotný HTTP protokol neumožňuje šifrování, k těmto účelům se používá TLS spojení nad TCP, označované jako HTTPS.

Protokol funguje tak, že nejdříve klient pošle serveru HTTP dotaz a server mu následně odpoví HTTP odpovědí. HTTP dotaz je ve formě prostého textu a v hlavičce nese typ požadavku (např. GET, SET, atd.) a označení požadovaného dokumentu. Mimo jiné se také v hlavičce uvádí server na který se dotazujeme, délka a typ obsahu požadavku a další.
Server následovně odpoví a v hlavičce je číslem předán výsledek požadované operace (např. 200 OK, 201 Created atd.), podle toho, jestli se dokument podařilo najít, či nikoliv.[1]

Příklad hlavičky odpovědi server aplikace při úspěšné operaci:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 61
```

HTTP/1.1

Tato verze HTTP definována v RFC2616 roku 1999 obsahuje oproti dřívější verzi (HTTP/1.0) více druhů požadavků v zájmu zajištění spolehlivé implementace všech jejích funkcí.[2] Ve verzi HTTP/1.0 byly definovány požadavky **GET**, **HEAD** a **POST**, v této verzi byly však přidány další jako např.: **PUT**, **DELETE** a další. V našem projektu jsou tyto požadavky také využity. Mimo jiné přišla také přišla možnost provozovat více WWW serverů na jedné adrese, přenos více souborů po sobě v jednom spojení a udržování spojení TCP.[1]

Implementace

K implementaci jsem použil jazyk C a vývojové prostření CLion. Operační systém, na kterém byl projekt zpracováván a překládan je Fedora 26 s nainstalovanou verzí gcc (GCC) **7.2.1** 20170915 (Red Hat 7.2.1-2). Překlad proběhl s parametry **-Wall**, **-Wextra** a **-pedantic**. Pro komunikaci po síti bylo využito BSD socketů. Projekt byl přeložitelný na serveru Merlin a také na něm proběhlo testování.

Klient aplikace

Aplikace nejdříve zkontroluje správnost zadaných argumentů a v případě, že některý argument chybí, nebo formát požadavku není správný, upozorní uživatele, ukončí se a nedojde k odeslání HTTP požadavku.
V případě, že jsou argumenty v pořádku, zapíše hlavičku s potřebnými informacemi a tělo s obsahem, je-li to třeba do bufferu, který je připraven k odeslání. Velikost tohoto bufferu je 4096 znaků a program nedovoluje překročit velikost tohoto bufferu.
V této fázi aplikace vytvoří socket, a naváže pomocí něj spojení se zadaným serverem. V případě, že dojde k chybě při vytváření socketu, navazování spojení, nebo při odeslání/příjmu dat je na to uživatel upozorněn a aplikace ukončena.
Po navázání spojení jsou odeslána data z bufferu a následně se čeká na přijetí odpovědi serveru. Po přijetí odpovědi aplikace volá funkce ze souboru *stringFunctions.c*, které přijatá data rozdělí na hlavičku a tělo HTTP protokolu.
Následně už je hlavička vypsána na stderr a tělo na stdout. Po vypsání je uvolněna alokovaná paměť a aplikace je ukončena.

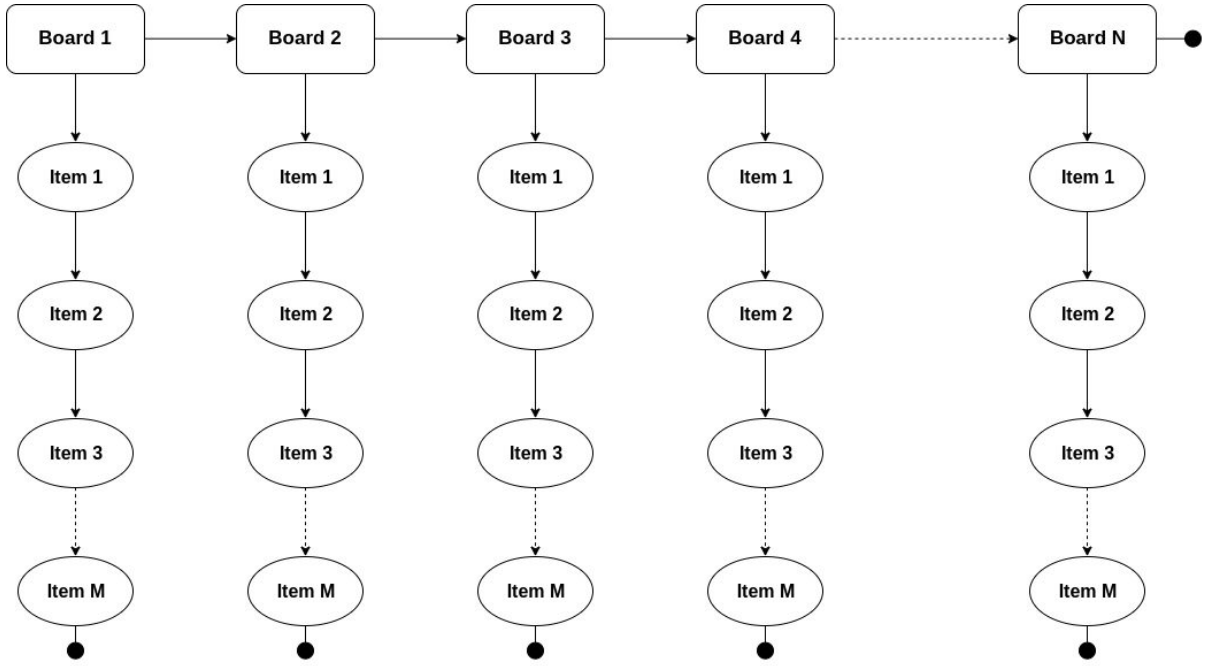
Server aplikace

Stejně jako klient aplikace, server nejdříve zkontroluje správnost zadaných argumentů. Následně se otevře socket a začne se naslouchat na zadaném portu, pokud by došlo k neúspěšné práci se socketem (např. zadaný špatný port), aplikace to nahlásí a ukončí se.
Server vytvoří ukazatel na začátek lineárního seznamu nástěnek, který je zatím prázdný a hodnota v ukazateli je NULL.
Jakmile se klient připojí k serveru a pošle svůj požadavek, server pomocí funkcí v souboru *stringFunctions.c* rozparsuje požadavek na požadovanou akci, popřípadě jméno tabulky a id. Pokud je to k dané akci potřeba, získá také tělo HTTP požadavku a zkontroluje v hlavičce, jestli se jedná o správný typ obsahu (Content-Type) a také jestli má správně udanou délku (Content-Length).
Pokud jsou všechny informace správné, tak na základě požadavku zjistí, jakou akci má udělat a následně již volá funkce pro práci se seznamem ze souboru *listFunctions.c*. Tyto funkce zajišťují práci se seznamem, ať už se jedná o výpis, vytváření, změnu nebo mazání.

Nakonec podle návratové hodnoty těchto funkcí zjistí, jestli byla požadovaná operace úspěšná a podle toho vytvoří hlavičku s příslušnou zprávou. Pokud se jednalo o akci, která to vyžaduje, přidá do hlavičky informace o typu a délce obsahu a do těla obsah samotný. Následně už pouze odešle vytvořený HTTP obsah, uvolní alokovanou paměť a čeká na dalšího klienta. Implementace serveru však neumožňuje přístup více uživatelů naráz.

Implementace seznamu nástěnek

Jedná se o jednosměrně vázaný lineární seznam, který nese jméno nástěnky, ukazatel na následující nástěnku v seznamu a ukazatel na seznam, který reprezentuje příspěvky v dané nástěnce.



Zajímavé pasáže

```
void intHandler()
{
  DeleteAllBoards(&head);
  close(my_sck);
  exit(0);
}
```

Toto je funkce v server aplikaci, která se zavolá po odchycení signálu CTRL+C, která zajistí korektní uzavření socketu a také vyčištění veškeré alokované paměti.

Funkčnost

Zde jsou přiloženy snímky obrazovky, které zobrazují přeložitelnost na serveru Merlin (obr. 1) a také na Fedoře (obr 2).

```
xchlub02@merlin: ~/ISA2$ ls
isaclient.c  isaserver.c  listFunctions.c  listFunctions.h  Makefile  stringFunctions.c  stringFunctions.h
xchlub02@merlin: ~/ISA2$ make
rm -f *.o *.out isaclient isaserver *~
gcc -Wall -Wextra -pedantic -o isaclient isaclient.c stringFunctions.c
gcc -Wall -Wextra -pedantic -o isaserver isaserver.c stringFunctions.c listFunctions.c
xchlub02@merlin: ~/ISA2$
```

```
[chlubnap@localhost ISA_Again]$ make
rm -f *.o *.out isaclient isaserver *~
gcc -Wall -Wextra -pedantic -o isaclient isaclient.c stringFunctions.c
gcc -Wall -Wextra -pedantic -o isaserver isaserver.c stringFunctions.c listFunctions.c
[chlubnap@localhost ISA_Again]$
```

Výpis Valgrindu u server aplikace.

```
==22333== HEAP SUMMARY:
==22333==     in use at exit: 0 bytes in 0 blocks
==22333==   total heap usage: 1,010 allocs, 1,010 frees, 51,916 bytes allocated
==22333==
==22333== All heap blocks were freed -- no leaks are possible
==22333==
```

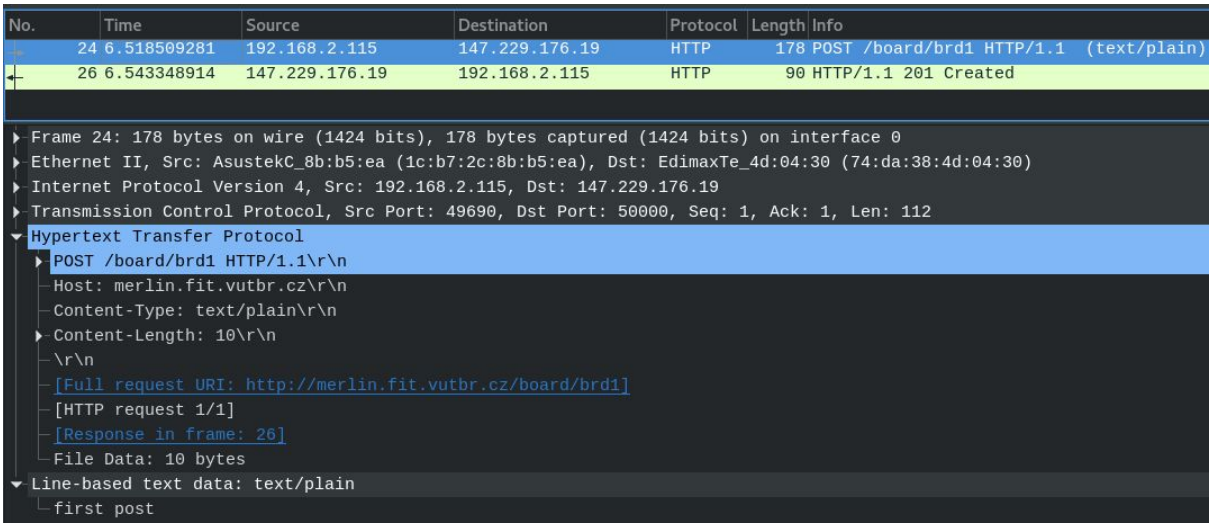
Testování

Přikládám také některé snímky obrazovky, které jsem pořídil při testování. Server aplikace byla spuštěna na Merlinovi a klient aplikace na mém počítači.

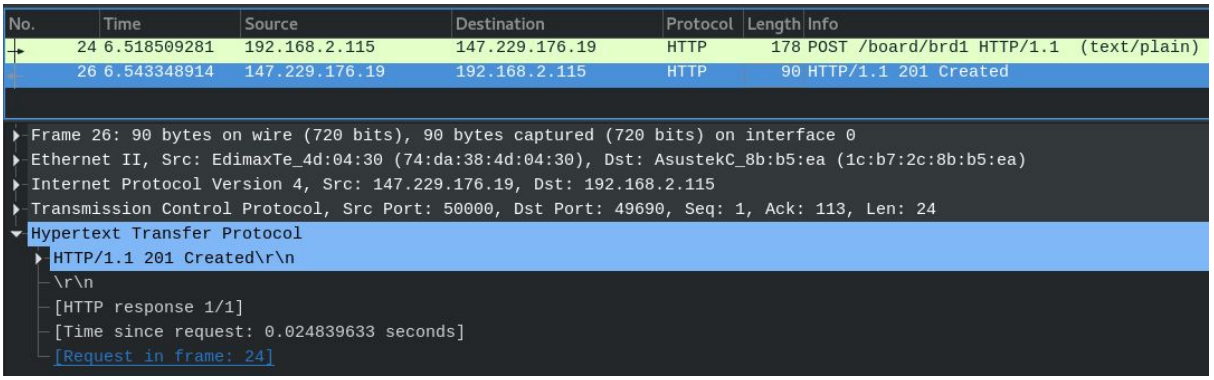
```
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 board add board5
HTTP/1.1 201 Created
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 boards
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 35
board1
board2
board3
board4
board5

[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 item add board2 "item number 1"
HTTP/1.1 201 Created
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 item add board2 "item number 2"
HTTP/1.1 201 Created
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 item add board2 "item number 3"
HTTP/1.1 201 Created
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 item add board2 "item \n\tnumber 4"
HTTP/1.1 201 Created
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 board list board 2
HTTP/1.1 404 Not Found
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 board list board2
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 77
board2
1. item number 1
2. item number 3
3. item number 3
4. item
        number 4

[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 item delete board2 1
HTTP/1.1 200 OK
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 item update board2 1
HTTP/1.1 400 Bad Request
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 item update board2 1 "Updated\tpost"
HTTP/1.1 200 OK
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 board list board 2
HTTP/1.1 404 Not Found
[chlubnap@localhost ISA_Again]$ ./isaclient -H merlin.fit.vutbr.cz -p 50000 board list board2
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 59
board2
1. Updated      post
2. item number 3
3. item
        number 4

[chlubnap@localhost ISA_Again]$
```

Pro ukázku jsem také nechal zachytit komunikaci pomocí programu WireShark:

Zachycený packet s požadavkem na přidání nového příspěvku do tabulky



Zachycený packet s odpovědí serveru na požadavek



Návod

Projekt je nejdříve nutno přeložit. To učiníme pomocí příkazu *make*, který přeloží obě aplikace a vytvoří dva spustitelné soubory, jejichž spuštění je popsáno níže. Pokud byste chtěli smazat spustitelné soubory, stačí zadat příkaz *make clean*.

Server aplikace

Server aplikace se spouští následovně:
./isaserver -p **<port>**, kde za **<port>** dosadíme číslo, které označuje port na kterém bude server naslouchat a čekat na příchozího klienta.

Příklad spuštění může být následující: ./isaserver -p 50000

Aplikaci ukončíme kombinací kláves CTRL a C, která ukončí program. Mimo jiné se může aplikace spustit s parametrem -h, který vypíše návod na spuštění aplikace.

Klient aplikace

Klient aplikace se spouští následovně:
./isaclient -H **<host>** -p **<port> <command>**
<host> - dosadíme adresu serveru, na kterém běží server aplikace.
<port> - dosadíme port, na kterém chceme aby komunikace probíhala a na kterém server naslouchá

Za **<command>** můžeme dosadit následující:

- boards - vypíše všechny nástěnky
- board add **<name>** - vytvoří nástěnku se jménem **<name>**
- board delete **<name>** - smaže nástěnku se jménem **<name>**
- board list **<name>** - vypíše obsah nástěnky se jménem **<name>**
- item add **<name> <content>** - přidá příspěvek do nástěnky se jménem **<name>** a textem **<content>**
- item delete **<name> <id>** - smaže příspěvek z nástěnky se jménem **<name>** na pozici **<id>**
- item update **<name> <id> <content>** - v nástěnce se jménem **<name>** změní text příspěvku na pozici **<id>** na nový text **<content>**

veškeré parametry předávané aplikaci jsou **case sensitive** a **<content>** se píše do dvojitých uvozovek (**“**text příspěvku**”**).
Ukázky spuštění aplikace jsou vidět na screenshotech v sekci testování.

Závěr

Obě dvě aplikace se podařilo naimplementovat a otestovat. Komunikace i operace nad seznamem fungují dle mého testování správně. Největší práce byla s parsováním obsahu HTTP protokolu, když jsem potřeboval získat jednotlivé parametry z hlavičky, jelikož jazyk C není pro práci s řetězci nejvíce přátelský. Také bylo potřeba celou hlavičku převést do lower case podoby a zbavit mezer, aby byla server aplikace připravena na čtení hlaviček s odlišným formátováním.
Také bylo potřeba vytvořit množství funkcí pro práci se seznamem a připomenout si, jak správně pracovat s ukazateli. Aby server aplikace nezanechávala alokovanou paměť, musel jsem si hlídat veškeré alokování a následně paměť korektně uvolňovat.
Implementace vytváření socketu a následné posílání dat nedělal větší problémy, jelikož jsem si to zkusil už v IPK.

Citace

1. Hypertext Transfer Protocol - Wikipedie. [online]. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol

2. Hypertext Transfer Protocol -- HTTP/1.1 - rfc2616. [online]. Dostupné z: https://tools.ietf.org/html/rfc2616