VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

APLIKACE PRO ZÍSKÁNÍ STATISTIK O SÍŤOVÉM PROVOZU

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE **TOMÁŠ DANIEL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MATĚJ GRÉGR, Ph.D.**
SUPERVISOR

BRNO 2024

Obsah

Kapitola 1

Úvod

Cílem semestrálního projektu bylo vytvoření konzolové aplikace *isa-top*, která bude zachytávat síťový provoz na uživatelem zvoleném rozhraní. Následně vypíše do konzole přenosové rychlosti pro jednotlivá zachycená spojení prostřednictvím knihovny *ncurses*, přičemž seznam bude obsahovat deset nejvíce komunikujících adres. Adresy mohou být řazeny dle počtu bajtů či počtu paketů za sekundu. Uživatel může při spouštění programu určit řadící metriku pomocí příznaku *-s*, nebo je použita výchozí metrika počtu bajtů/s.

Kapitola 2

Návrh a implementace aplikace

Návrh aplikace odpovídá využití objektově orientovaných konceptů a možnostem, které implementace v jazyce C++ nabízí. Aplikace je členěna na logické celky v podobě tříd, které jsou rozděleny a implementovány v samostatných souborech. Orchestrace poté probíhá v těle hlavní funkce *main()* umístěné v souboru *main.cpp*.

Tuto kapitolu jsem rozčlenil do sekcí, které se postupně věnují popisu jednotlivých tříd a částí programu.

2.1 Třída *NetworkData*

Tato třída je implementována v souborech:

- *NetworkData.cpp*
- *NetworkData.h*

a jejím primárním účelem je práce se síťovým tokem a následná extrakce dat prostřednictvím knihovny *libpcap*[1] ze zadaného síťového rozhraní. Konstruktor této třídy přijímá jediný argument a tím je název cílového rozhraní na kterém má třída provádět svou činnost. Ještě před samotným zahájením je ověřena validita zadaného rozhraní pomocí metody *NetworkData::validateInterface()*.

Následuje invokace metody *NetworkData::startCapture()* z hlavní funkce programu *main()*, která zahájí zachytávání síťové komunikace v samostatném *std::jthread*[2] vláknu metodou *NetworkData::capturePackets()*. Tato metoda inicializuje handler pro odchytávání provozu na zvoleném rozhraní. Zachycené pakety jsou následně zpracovány funkcí *handlePacket()*, která v nich v závislosti na verzi Internetového protokolu (*IP*) extrahuje z hlaviček paketů zdrojovou a cílovou IP adresu a v případě užití protokolů *TCP* či *UDP* rovněž i hodnoty zdrojových a cílových portů. Takto extrahované hodnoty komunikace jsou uloženy do privátní mapy této třídy, ze které jsou prostřednictvím veřejné metody *getCurrentData()* periodicky získávána data pro terminálové zobrazení.

[1] Oficiální stránky knihovny: https://www.tcpdump.org/
[2] Bližší popis třídy: https://en.cppreference.com/w/cpp/thread/jthread

2.2 Třída *Outputter*

Tato třída je implementována v souborech:

- *Outputter.cpp*
- *Outputter.h*

a slouží pro zobrazování dat periodicky získávaných ze třídy *NetworkData*. Pro zvýšení přehlednosti a estetičnosti terminálového výstupu je použita knihovna *ncurses*[3]. Konstruktor této třídy přijímá jediný argument, který určuje metodiku podle které budou výsledky řazeny, zda-li dle počtu paketů nebo bajtů za sekundu.

Mapa obsahující zpracovaná síťová data ze třídy *NetworkData* je přijata metodou *Outputter::processData()*, kde jsou převedeny do formátu *std::vektor*[4] za účelem závěrečného řazení. Takto seřazená, a v případě počtu záznamů > 10 rovněž oříznutá, data jsou předána metodě *Outputter::showData()* a ta v souladu s výstupním formátem dat ze zadání nastaví příslušné sloupce v terminálu a data zobrazí. Aby bylo dodrženo zadání, konkrétně převody mezi předponami *kilo (K)*, *mega (M)* a *giga (G)*, jsou získané hodnoty počtu přenesených bajtů a paketů metodou *Outputter::convertValue()* příslušně formátovány a zaokrouhleny na jedno desetinné místo.

2.3 Třída *ProgramException*

Tato třída je implementována v souborech:

- *CustomException.h*

a jedná se o implementaci vlastního typu výjimky, který dědí od standardní třídy výjimek *std::exception*[5]. Důvodem vzniku bylo jednoznačně odlišit případné výjimky generované mým programem a ty korektně zpracovat.

2.4 Soubor *main.cpp*

Tento soubor obsahuje hlavní funkci programu *main()*. Tato funkce na počátku zpracovává uživatelem zadané argumenty spuštění, v případě nevalidní nebo chybějící hodnoty je vyvolána výjimka *ProgramException* a program je ukončen s chybovou návratovou hodnotou (-1). Tyto zadané hodnoty jsou během zpracování ukládány do mapy a následně použity při tvorbě instancí tříd *NetworkData* a *Outputter*. Poté již následuje zahájení sběru a zpracování síťového provozu zavoláním metody *NetworkData::startCapture()* a periodické předání těchto dat třídě *Outputter* prostřednictvím metody *Outputter::processData()*.

2.5 Soubor *Includes.h*

Hlavičkový soubor, ve kterém jsou centralizovány přidávání veškerých potřebných knihoven a tříd, které program vyžaduje.

[3] Bližší informace k nalezení zde: https://invisible-island.net/ncurses/ncurses.html
[4] Detaily: https://en.cppreference.com/w/cpp/container/vector
[5] Popis třídy: https://en.cppreference.com/w/cpp/error/exception

Kapitola 3

Základní informace o programu

Program je implementovaný v jazyce C++, konkrétně ve standardu C++20[1]. Součástí tohoto standardu je i podpora formátování textu pomocí *<std::format»*, který je v tomto programu použitý, je **proto nutné pro překlad použít kompiler GCC[2] ve verzi >= 13**.

[1] Přehled novinek standardu: https://en.cppreference.com/w/cpp/20
[2] Oficiální stránky: https://gcc.gnu.org/

Kapitola 4

Použití programu

Použití programu je v souladu se zadáním, tedy pro překlad a vygenerování spustitelného souboru *isa-top* je terminálový příkaz *make* a pro odstranění *make clean*.

Při spuštění programu je možné specifikovat troje hodnoty:

- *-i INTERFACE* ... kde *INTERFACE* je název rozhraní na kterém má program provádět svou činnost
- *-s* [b|p] ... určuje použitou metodu pro řazení výstupu, *"b"* pro řazení dle počtu přenesených bajtů a *"p"* dle počtu paketů
- *-h* ... pro zobrazení nápovědy ke spuštění programu

vzhledem k činnosti programu je **nutné ho spouštět v privilegovaném režimu**.

4.1 Příklad spuštění

Výstupem při spuštění programu **sudo ./isa-top -i eth0** může být[1]:

Src IP:port	Dst IP:port	Proto	Rx		Tx	
			b/s	p/s	b/s	p/s
172.19.251.99:56720	77.75.79.195:443	tcp	14.7K	182.0	2.6M	215.0
172.19.251.99:47699	142.251.37.100:443	udp	43.3K	170.0	1.0M	852.0
172.19.251.99:36028	77.75.77.195:443	tcp	6.0K	57.0	225.3K	51.0
172.19.251.99:53796	77.75.76.30:443	tcp	45.8K	125.0	185.0K	104.0
172.19.251.99:37888	77.75.77.222:443	tcp	8.9K	49.0	140.8K	48.0
172.19.251.99:59704	142.251.37.99:443	udp	4.9K	33.0	87.7K	75.0
172.19.251.99:59394	77.75.76.20:443	tcp	30.1K	26.0	6.6K	30.0
172.19.251.99:39512	77.75.79.115:443	tcp	1.8K	13.0	30.5K	12.0
172.19.251.99:35998	77.75.77.195:443	tcp	2.7K	23.0	21.3K	19.0
172.19.251.99:56708	77.75.79.195:443	tcp	2.2K	20.0	21.7K	17.0

kde je možné vidět jednotlivé komunikující adresy pomocí daného protokolu ve sloupci *Proto* a přenosové rychlosti (sloupce *b/s*) a počty přenesených paketů (sloupce *p/s*).

[1] Pro demonstrační účely byly zvoleno užší mezery mezi sloupci než v případě reálného použití

Kapitola 5

Testování

python skripty pro generov8n9 UDP a ICMP provozu a IPv6

V následující kapitole se zaměřím na rozebrání a popsání jednotlivých použitých testovacích metodik, které jsem aplikoval při ověřování správnosti mojí implementace.

5.1 Popis a výsledky testování

5.1.1 Obecné testování

Obecné testování probíhalo jako první v pořadí, v jeho rámci jsem chtěl ověřit jednak stabilitu programu a druhak identifikovat případné paměťové úniky.

Stabilitu a funkčnost programu jsem ověřoval v rámci subsystému *WSL*[1] s nainstalovaných operačním systémem *Kali linux*[2] a to tak, že jsem v jednom terminálovém okně zapnul svůj program a ve druhém webový prohlížeč, ve kterém jsem procházel nahodilé známé stránky a sledoval jsem, zda-li program tento provoz zachytí a korektně zobrazí. Volba tohoto testovacího scénáře vycházela z faktu, že provoz na webových stránkách je v drtivé většině případů prováděn přes protokol TCP, což mi pro test zajistilo nemalé množství síťového provozu.

Pro kontrolu nakládání s pamětí ze strany programu jsem použil nástroj *Valgrind*[3], který odhalil pouze paměťové úniky spojené s knihovnou *ncurses*[4].

5.1.2 Testy jednotlivých protokolů

Úkolem této části testování bylo vyzkoušet schopnost programu zachytávat komunikaci rozličných protokolů, schopnost zachytávat protokoly *TCP* a *UDP* ověřily už výše uvedené testy. Pro protokol *ICMP* jsem využil softwarovou utilitu *ping*, která generuje *ICMP ECHO_REQUEST* síťový provoz.

[1] Dokumentace: https://learn.microsoft.com/en-us/windows/wsl/
[2] oficiální stránky: https://www.kali.org/
[3] Stránky nástroje: https://valgrind.org/
[4] Zdůvodněno zde: https://invisible-island.net/ncurses/ncurses.faq.html#config_leaks

5.1.3 Testy pro ověření správnosti počtu přenesených paketů

Program prezentuje, krom počtu přenesených bajtů, rovněž i údaj o počtu přenesených paketů, pro ověření správnosti tohoto údaje sem opět použil *ping*, kde jsem pomocí přepínače *-c COUNT* nastavil počet vygenerovaných *ICMP* zpráv a při ukončení této utility se na závěr vypíše i počet odeslaných a přijatých paketů, tuto hodnotu jsem porovnal s mými hodnotami a tím ověřil jejich správnost.

Literatura