VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MONITOROVÁNÍ FUNKČNOSTI BĚŽÍCÍCH APLIKACÍ

MONITORING OF THE FUNCTIONALITY OF RUNNING APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ANDREI PAPLAUSKI**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RUDOLF ČEJKA**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Paplauski Andrei**
Program: Informační technologie
Název: **Monitorování funkčnosti běžících aplikací**
Monitoring of the Functionality of Running Applications
Kategorie: Operační systémy

Zadání:

1. Seznamte se principem fungování reálného systému řízení letového provozu.
2. Identifikujte situace zamrznutí jednotlivých komponent systému.
3. Navrhněte různé metody sledování jednotlivých komponent systému tak, aby bylo možné identifikovat problémy v běhu komponent.
4. Implementuje možnost simulace chyb vybraných komponent systému.
5. Implementujte prototyp pro sledování chyb pomocí navržených metod
6. Otestujte a zhodnoťte vytvořený systém

Literatura:

- Operační systém Linux: https://www.linux.org/
- X Window System: https://www.x.org/

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Čejka Rudolf, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 30. října 2018

Abstrakt

Cílem této práce je vytvoření systému, který bude sloužit k monitorování běhu aplikace s zaměřením na sféru řízeni letového provozu. Práce analyzuje problémy monitoringu aplikaci v dané sféře a navrhuje efektivní řešeni, které mohlo by byt užitečné pro techniky nebo administrátoři různých systémů. Systém se skládá z klientů, které běží na řídicích stanicích, sbírají určitá data a odesílají jich na server pro následující analýzu a zobrazení. Sbíraje se standardní data o systému a procesu, jako použitá paměť, využití procesoru, volné místo na disku atd. Ale kromě toho klient umí dělat screenshot obrazovky a analyzovat jak se změnil obrázek vůči předchozímu stavu. Vytvořený systém lze použit pro monitorovaní běhu různých aplikací běžících na různých počítačích v společném systému.

Abstract

The aim of this work is to create a system that will serve to monitor running applications with a focus on the sphere of air traffic control. The thesis analyzes problems of monitoring the application in the given sphere and proposes an effective solution that could be useful for technicians or administrators of different systems. The system consists of clients running on the control stations, collecting certain data and sending them to the server for the next analysis and display. Standard system and process data are collected, such as memory usage, CPU usage, free disk space, etc. In addition, the client can do a screenshot and analyze how the image has changed from the previous state. The created system can be used to monitor different applications running on different computers in a single system.

Klíčová slova

Linux, monitorovaní, klient, server, c++, letový provoz.

Keywords

Linux, monitoring, client, server, c++, air trafic.

Citace

PAPLAUSKI, Andrei. *Monitorování funkčnosti běžících aplikací.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Rudolf Čejka

Monitorování funkčnosti běžících aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rudolfa Čejky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Andrei Paplauski
13. května 2019

Obsah

Kapitola 1

Úvod

Sledovaní a analýza běhu programů nebo produktu obecně – je velice důležitá část ověření jeho správností a funkčnosti. I když technologie je něco podstatného, to neznamená, že je to neomylné. Poruchy, které způsobují někdy kritické situace, se mohou kdykoli objevit. Proto všude, kde je důležitá počítačová infrastruktura, bude nutné kontrolovat její správné fungování tak, aby případná chyba neovlivnila službu poskytovanou uživatelům. Je nutně aby administrátor nebo technik si uvědomoval, co se děje s určitým počítačem, nebo programem na tom počítači. Proto existují různá monitorovací prostředky, které pomáhají sbírat a analyzovat informace z počítačů.

Obzvlášť sledování aplikaci ma vyznám ve sféře řízení letového provozu, kde od spolehlivého běhu programu závisí bezpečnost lidí a letadel. Obvykle řidiče v letovém informačním středisku mají hodně důležitých úkolů, jako poskytování letové informaci, komunikace s piloty a stanici, sledováni letového prostoru atd. Při takové práci občas dispečer muže přehlédnout, že nějaký program se chová ne tak jak ma. Také existují chyby které nejde snadno zjistit uživateli softwaru, jako například nedostatek místa na disku nebo nadměrné využití zdrojů počítače.

V dnešní době existuje dost monitorovacích prostředků. Každý má nějaké výhody a nevýhody. Některé sledují stav systému z vetším důrazem na síťovou komunikaci, některé sledují jen systémové volaní programu, jiné – celkový stav systému bez ohledu na běžící procesy. Nevýhody takových prostředků můžou byt v tom, že oni nejsou projekty s otevřeným zdrojovým kódem, nebo oni potřebují přímé zasahování do zdrojových kódu aplikaci, což není vždycky možné. A proto jsem rozhodl, že vytvořím systém, který bude malý, jednoduchý pro použití, ale při tom efektivní a užitečný.

V následující kapitole 2 je stručný úvod do historie letového provozu v České Republice a proč systémy monitorovaní jsou tak důležité v teto sféře.

Dal v kapitole 3 jsou analyzované některá z už existujících prostředku pro sledovaní běhu aplikaci nebo systému, jejích slabé a silné strany. Také v kapitole je informace o tom, jak se řeší monitorování ve sféře řízení letového provozu, a jak tato situace mohla by byt zlepšena.

V kapitole 4 jsou popsané všechny použité knihovny třetích stran, které napomáhali sbíraní dat a jejich přenosu mezi stanici a serverem. Proč zrovna byli použité tyto knihovny, principy fungovaní a některé důležité aspekty jejích použití v projektu.

Pak v kapitole 5 je popsaná základní struktura projektu a proveden návrh monitorovacího nástroje s popisem jeho klíčových vlastností s rozdíly oproti existujícím řešením. V další kapitole 6 je postup implementace programu, jeho zajímavé prvky, datové typy, důležité funkce. Vysvětleny společné části klienta a serveru, stejně jako i jejích rozdíly.

Poslední kapitola 7 se popisuje proces a výsledky testovaní jak celého systému a jeho integraci, tak i samotných prvku. Také ohodnocen výsledný stav projektu a některeře ze zjištěných problémů.

Kapitola 2

Systém řízení letového provozu a současné monitorovací prostředky

V teto kapitole je popsaná historie letového provozu od minulosti do současného stavu. Jaké vlastnosti on ma v dnes a jaké problémy v teto oblasti můžou existovat.

2.1 Letový provoz a systémy jeho řízení

Management letového provozu se skládá ze dvou systémů, systém řízení letového provozu (ŘLP) a systém řízení toku letového provozu (ŘTLP). ŘLP zahrnuje monitorování a kontrolu letadel, aby zajistit, že oni budou bezpečně, odděleně a efektivně pracovat. Obvykle se snaží identifikovat a vyřešit problémy, které by mohly vzniknout v příštích dvaceti minutách. ŘTLP je strategičtější: zpožďuje lety nebo upravuje trajektorie až několik hodin předem. Takže ŘLP nemusí pracovat s řadou letů, které je obtížně efektivně oddělit. Jinými slovy se ŘTLP pokouší vyhnout překročení úrovně vzdušného prostoru a kapacity letišť, které jsou nastaveny tak, aby zajistily bezpečné ŘLP. ŘTLP rovněž usiluje o spravedlivé využívání dostupné kapacity a poskytnutí provozovatelům letů flexibilitu chovat se podle svých preferencí nebo obchodních modelů.

Řízení letového provozu v České Republice zahrnuje několik typů služeb, a to jsou [22]:

1. Služba řízení letového provozu (ATC), do níž patří:
   - Oblastní služba řízení
   - Přibližovací služba řízení
   - Letištní služba řízení / Služba řízení na odbavovací ploše
2. Letová informační služba (FIS)
3. Pohotovostní služba (ALRS)
4. Ohlašovna letových provozních služeb

2.2 Historie řízení letového provozu

Od počátku dvacátých let ve vyspělých státech vznikala a rychle se rozvíjela letecká doprava. Tím začalo komerční využívání vzdušného prostoru. Ovšem podíl letecké dopravy na celkových dopravních výkonech byl velmi malý. Omezené možnosti byly dány úrovní tehdejší letadlové techniky, malou nosností, doletem a spolehlivostí letadel. Vyšší rychlost a tedy úspora času proti pozemní dopravě se však významně uplatňovala při dopravě pošty, denního tisku i jednotlivých pasažérů. K narušování pravidelnosti letecké dopravy přispívala závislost na počasí i na charakteristikách a stavu letišť.

S časem stalo zřejmě, že vzdušný prostor je důležitým zdrojem národního bohatství každého státu. Proto se rozvinula široce založená teritoriální spolupráce zemí, v jejichž vzdušném prostoru je intenzivní letový provoz. Vznikli opatření pro účelné uspořádání a řízení toku letového provozu, s cílem zajistit jeho plynulost a pravidelnost, při zachování nebo zlepšení jeho bezpečnosti.

Velmi důležitým aspektem rozvoje letecké dopravy byl a je proces zvyšování její bezpečnosti. Letecká doprava byla na počátku svého vývoje dopravou nejméně bezpečnou. Zatímco v letech 1925-29 připadalo na 1 miliardu osobokilometrů 28 mrtvých, v období 1945-1950 tento počet klesal ze 3 na 2 a v letech 1991-1995 dosáhl hodnoty od 0,5 do 0,3. Soustavným a nepřetržitým úsilím všech zúčastněných složek bylo dosaženo toho, že v současnosti je letecká jednoznačně nejbezpečnější mezi všemi ostatními druhy dopravy. Rostoucí objem letového provozu a efektivní využívání disponibilní kapacity vzdušného prostoru a letišť způsobuje, že nároky na tyto služby progresivně rostou. To vede k nezbytnosti budovat rozsáhlé řídicí, sledovací, komunikační a informační systémy, které patří mezi nejsložitější, nejrozsáhlejší a nejnákladnější systémy v celém odvětví dopravy.[21]

2.3 Problémy bezpečnosti v sféře řízení letového provozu

Během rušné letecké činnosti je koncentrace a stres kontrolérů tak těžké, že jejich chyby jsou důležitým zdrojem nebezpečí pro letu. Proto je velmi důležité aby aplikace, které používají dispečeři fungovali stabilně a spolehlivě. Pro řídicí jednotky existuje několik sad systémů řízení vzduchu, pomocných a příkazových systémů. Většina těchto systémů má společné charakteristiky, jako je pracovní stanice s vysokým rozlišením s vysokou diferenciálností, přátelské grafické uživatelské rozhraní a funkce, jako je zpracování radarových dat, zpracování letových údajů, automatické závislé sledování a detekce konfliktů a varování. Tyto systémy jsou velice složité součásti jediného systému, který používají kontroléry ve své práci, a proto monitorovaní jejích běhu hraje prvořadou roli pro zajištění bezpečnosti letišť a letů.

Kapitola 3

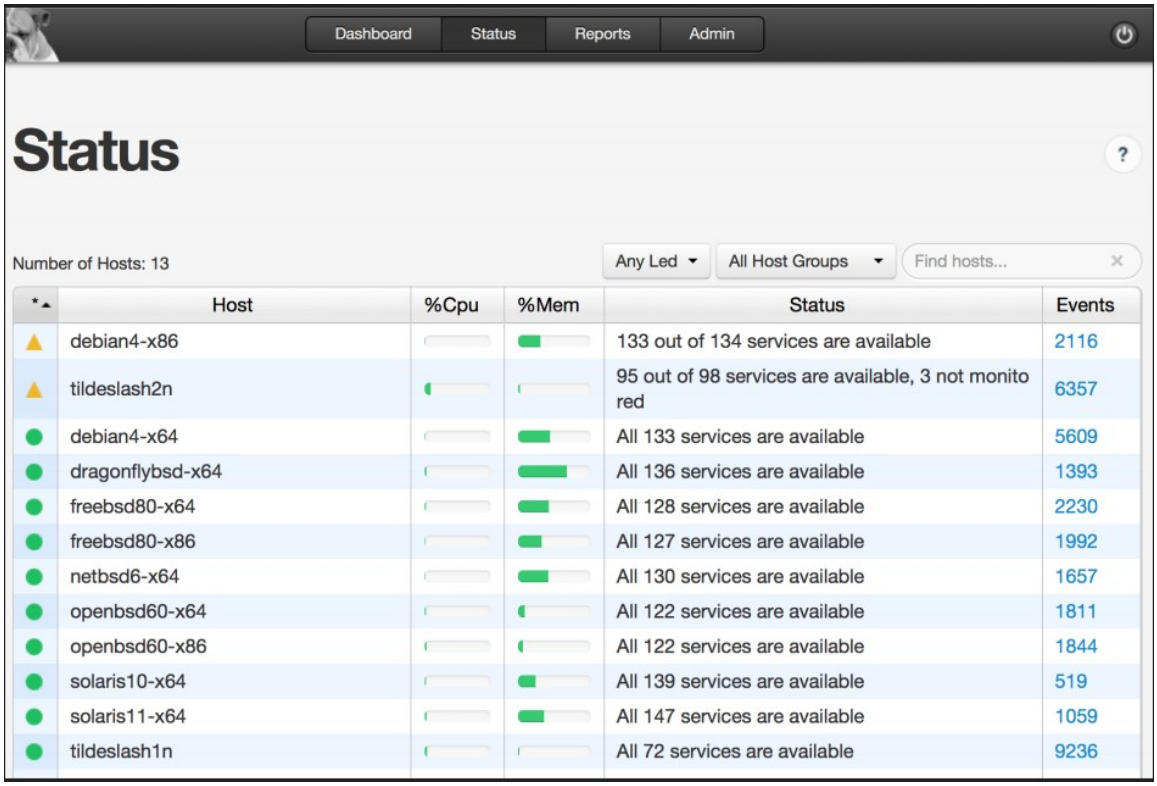
Existující monitorovací prostředky

V teto kapitole popsané některé z analyzovaných knihoven nebo aplikaci pro monitorovaní běhu programu. Také vysvětleno jaké mají nevýhody, případně proč jejích použití v systémech řazení letového provozu muže byt komplikované.

3.1 M/MONIT

M/Monit je systém pro automatické řízení a proaktivní sledování systémů informačních technologií. M/Monit může sledovat a spravovat distribuované počítačové systémy, provádět automatickou údržbu a opravy a realizovat smysluplné příčinné akce v chybových situacích.[8]

Aplikace pracuje podle principu jeden server a vice klientů. Kde na každém systému, který potřebuje sledování běží klient, sbíraje data o systému a aktivních službách, a odesílá přes šifrovaný kanál na server. Tam informace se zpracuje, analyzuje a zobrazuje pomoci webového rozhraní. Zobrazováni je reprezentována různými způsoby. Nejobecnější je tabulka se všemi sledovanými systémy a informaci o nich. Příklad takové tabulky je na obrázku 3.1. Další způsob zobrazení je analytické grafy různých typů: sloupcové, kruhové, lineární diagramy jsou používány podle kontextu. Také M/Monit zobrazuje aktuální chyby a varování, které nemůže vyřešit samostatně. To muže byt chyba při startu servisu, nemožnost načíst souborový systém, nebo dlouhá doba odpovědí klienta. Navíc monitorovací program umí pracovat s databázovými servery, kde uchovává záznam událostí a stav sledovaných systémů. On umí pracovat s takovými systémy řízení báze dat jako MySQL, PostgreSQL and SQLite.



Obrázek 3.1: M/Monit tabulka sledovaných systémů (převzato z https://mmonit.com/documentation/mmonit_manual.pdf).

M/Monit vice zaměřený na monitorovaní velkého množství systémů, a celého stavu daných systémů, než na monitorování stavu jedné aplikací. Proto se monitorují většinou jenom obecné věcí jako procesorový čas a použita paměť. Navíc tento produkt je komerční bez otevřeného zdrojového kódu a jeho podniková licence může byt dostatečně drahá pro některé firmy.

3.2 Systemtap

Systemtap je nástroj, který umožňuje vývojářům a správcům systému psát a znovu používat jednoduché skripty, aby hlouběji prozkoumali činnost živého systému Linux. Data mohou být extrahována, filtrována a shrnuta rychle a bezpečně, aby bylo možné diagnostikovat složité výkonové nebo funkční problémy.[5]

Na obrázku 3.2 je implementován jednoduchý sledovací script, který sleduje veškerou síťovou komunikaci protokolu IPv4 v systému. Při jeho ukončení se vytiskne seznam, který udává počet paketů odeslaných podle zadané adresy IP adresy/cílové adresy IP a celkový počet bajtů odeslaných mezi párem. Seznam je řazen od největšího k nejmenšímu počtu paketů mezi páry zdroj/cíl.

Některé pozoruhodné funkce jazyka SystemTap jsou:

- Kód je přeložen do jazyka **C**, kompilován do modulu jádra a načten.
- Středníky na konci výrazu jsou volitelné a obecně ve skutečnosti znamenají prázdný výraz.

```
#! /usr/bin/env stap


global allPackets

probe begin
{
       print("Start collecting data")
}

probe netfilter.ipv4.pre_routing
{
       allPackets[saddr, daddr] <<< length
}

probe end
{
       print("\n")
       foreach([saddr, daddr] in allPackets-)
       {
              printf("from %20s to %20s => bytes: %d, packets: %d \n",
                        saddr, daddr,
                        @sum(allPackets[saddr, daddr]),
                        @count(allPackets[saddr, daddr]))
       }
}
```

Obrázek 3.2: Příklad skriptu pro monitorovaní síťové komunikaci napsaném v skriptovacím jazyce SystemTap.

- Odvození typu (řetězec, číselné) se provádí automaticky v době kompilaci.
- Podporuje asociativní pole, implementované jako hash tabulky, jejichž maximální velikost je pevná a nastavena při spuštění skriptu. Asociativní pole musí být globální.
- Podporované řídící struktury *if-then-else, while, for, foreach, break, continue.*
- Globální proměnné použité v sondě jsou při vstupu do jiné sondy automaticky zablokovány.
- Veškerá paměť je při inicializaci skriptu přidělena jednou, a proto nejsou možné úniky paměti.
- Pokud je operátor příliš dlouhý nebo příliš hluboko do rekurze, detekuje se a skript se zastaví.
- Přimo ve skiptech lze používat kod v jazyce **C**.

SystemTap nevyžaduje ruční vkládání sond do kódu nebo aplikací jádra. Však vyžaduje náročné znalosti zdrojového kódu sledované aplikaci. Skripty SystemTap jsou přeloženy do

jazyka C a načteny do jádra jako moduly. Tato výhoda vyrovnává dlouhý kompilační čas. Skriptovací jazyk SystemTap je poměrně silný nástroj pro monitorování aplikací, ale není optimálním řešením tohoto problému.

3.3 LTTng

V teto sekci je popsán jeden z velice populárních softwaru pro sledovaní aplikaci a systémového jádra LTTng. Historie vzniku a vývoje programu. Jaké technologii podporuje, jak probíhá komunikace mezi jeho komponenty. A jaké má nevýhody při použiti.

V roce 1999 začal zaměstnanec IBM Karim Yaghmour pracovat na projektu LTT (Linux Trace Toolkit). LTT byl založen na následující myšlence: staticky instrumentovat nejdůležitější fragmenty v kódu jádra a získat tak informace o provozu systému[19]. O několik let později, tento nápad byl vyvinut Mathieu Denoye jako součást projektu LTTng (Linux Tracing Tool New Generation). První vydání LTTng proběhlo v roce 2005. A od toho roku až do dneška Mathieu Denuaye neustále pracuje na zlepšování LTTng. Poslední stabilní verze na moment napsaní teto paci je 2.10. [20]

Tento odstavec byl převzaty z [13]. LTTng je softwarová sada nástrojů s otevřeným zdrojovým kódem, která může být použita pro současné sledování jádra Linuxu, uživatelských aplikací a uživatelských knihoven.

LTTng se skládá z:

- Moduly jádra pro sledování jádra Linuxu.
- Sdílené knihovny pro sledování uživatelských aplikací napsaných v **C** nebo **C++**.
- **Java** balíčky pro sledování **Java** aplikací, které používají *java.util.logging* nebo *Apache log4j 1.2*.
- Balíček **Python** pro sledování aplikací **Python**, které používají standardní protokolovací balíček.
- Modul jádra pro sledování **shell skriptů** a dalších uživatelských aplikací bez vyhrazeného mechanismu instrumentace.
- Démony a nástroj příkazového řádku, *lttng*, pro ovládaní LTTng sond.

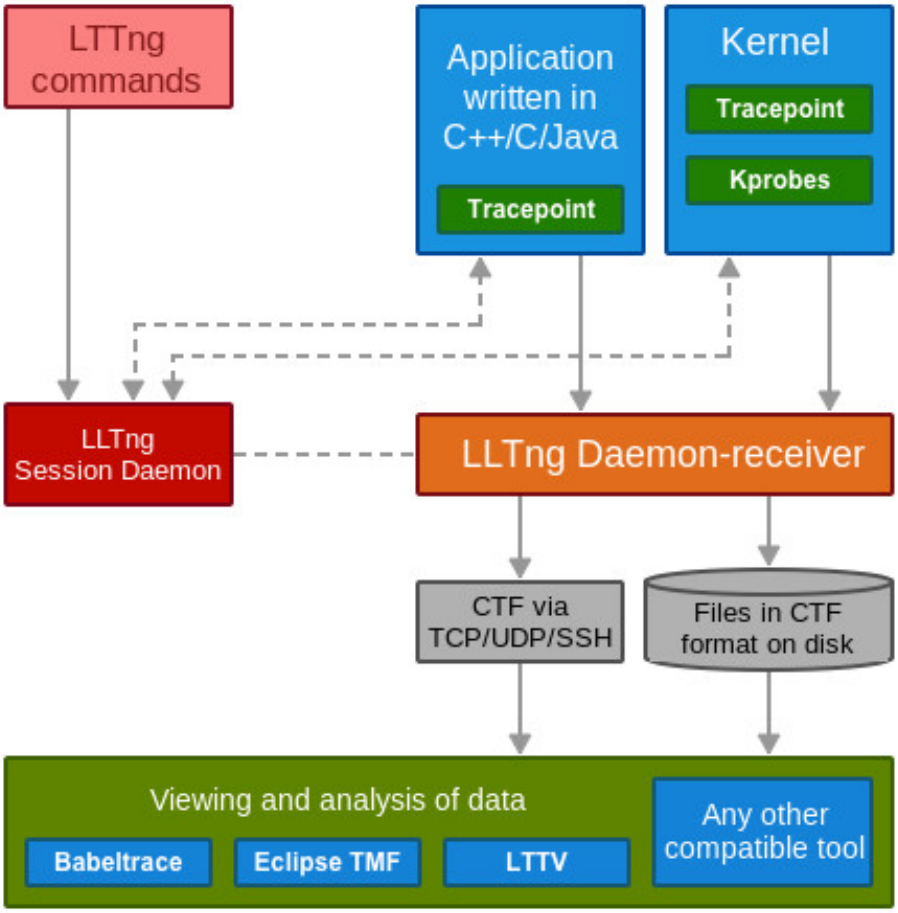
LTTng je obsažen v repozitářích nejmodernějších linuxových distribucí a můžete jej nainstalovat standardním způsobem pro konkretní system. LTTng se skládá ze dvou části *lttng-tools* a *lttng-modules*.

Balíček *lttng-tools* obsahuje následující nástroje:

- babeltrace – nástroj pro prozkoumávání výstupů sledovaní v CTF (Common Trace Format).
- lttng-sessiond – démon pro kontrolu trasování.
- lttng-consumerd – démon, který sbírá data a zapisuje je do kruhové schránky.
- lttng-relayd – je démon, který přenáší data po síti.

Balíček *lttng-modules* obsahuje mnoho modulů jádra, které jsou v interakci s vestavěnými mechanismy trasování a profilování.

Na obrázku 3.3 popsán základní proces komunikací komponent systému LLTng. Také na něm je zobrazeno jaké typů sledovacích démonů, jaké funkce mají a s kým komunikují. Jakou cestou se informace dostava z aplikaci do zobrazovacího serveru. Jaké typy komunikaci se při tom můžou byt použity. A pomoci čeho se sleduje jádro systému.



Obrázek 3.3: Proces interakce všech složek LTTng.

Základem monitorovaní LTTng jsou tak zvané **událostí**. LTTng sleduje informace o událostech v prostoru jádra i v uživatelském prostoru. Nejobecnější případy využiti tohoto softwaru jsou: analýza meziprocesorové komunikace v systému, analýza interakce aplikací v uživatelském prostoru s jádrem, měření času stráveného jádrem na servisních aplikacích, analýza vlastností systému při vysokém zatížení. Ale pro monitorovaní procesy, měli by byt udělány změny přímo v kódu aplikaci, odkud se budou posílat užitečná data. A to je hlavní nevýhodou toho softwaru.

3.4 DTrace

DTrace je určena pro dynamické sledování jádra systému a aplikací v reálném čase, zejména za účelem jejich profilování a ladění [1]. Původně vznikla pro operační systém Solaris ještě v roce 2005. A z licenčních důvodu měla problémy s portováním do operačních systémů Linux. DTrace má schopnost sledovat provoz aplikací ze zdrojového kódu, přes systémové knihovny, přes systémová volání až do jádra. Tato viditelnost umožňuje zjistit a kvantifikovat kořenovou příčinu problémů (včetně problémů s výkonem), a to i v případě, že je vnitřní ovladač ovladače jádra nebo něco jiného než hranice kódu aplikace[6].

Ve srovnání s podobným softwarem má DTrace tyto výhody:

- V mnoha systémech je DTrace primo součásti operačního systému, není potřeba nic instalovat.
- DTrace se před spuštěním trasování nezpomaluje na rozdíl od některých jiných monitorovacích systémů.
- Je k dispozici velice podrobná dokumentace.
- Funkcionalita vypadá pohodlněji a bohatší než u analogů.
- Sondy jsou stabilní a zdokumentované přímo pro vše události v systému - ve stylu navázaného TCP spojení, je TCP spojení uzavřeno atd.

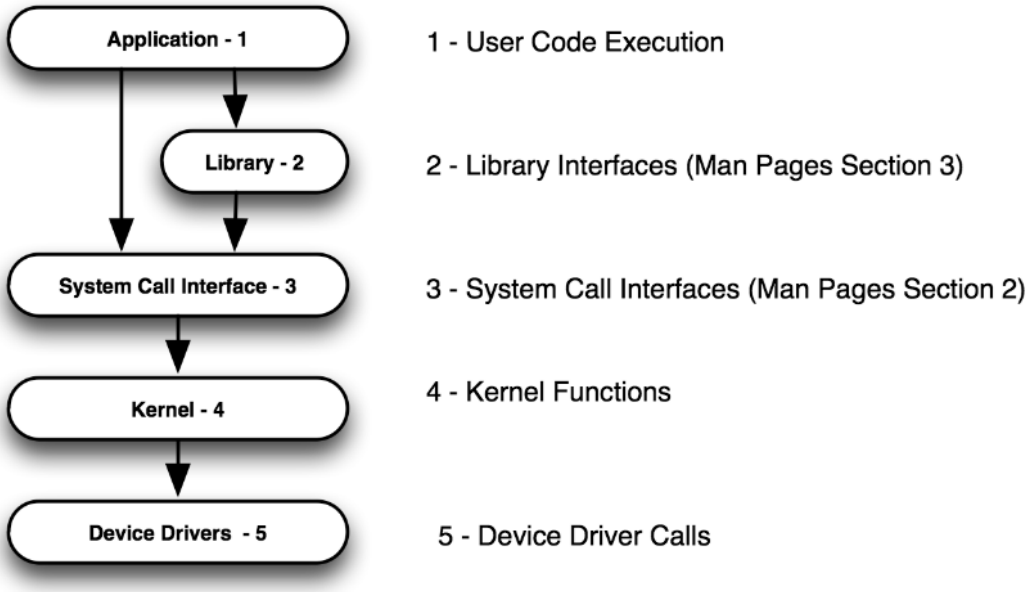
Jedna z důležitých vlastnosti DTrace je tak zvané *dynamické trasovaní*. Hlavní rozdíl mezi dynamickými trasami a jinými metodami analýzy jádra – schopnost vložit vlastní kód do aktivního jádra, která umožňuje poskytovaní velkého množství událostí a větší flexibilitu při jejich zpracování. Logika dynamického trasovaní je jednoduchá: skript je vytvořen v jazyce podobném jazyku **C** (v DTrace skripty mají příponu .d) a převedeny do binárního kódu cílové architektury, v DTrace se to provádí pomocí kompilátoru jazyka D (neplést s jazykem D od Digital Mars). Výsledný kód se přenáší do adresního prostoru jádra.

Zpracování informací získaných ze senzorů se provádí v modulu jádra DTrace. Definice každého senzoru se skládá ze čtyř prvků oddělených dvojtečkou. Obecně vypada takto:

poskytovatel:modul:funkce:jméno

- Poskytovatel: Model jádra DTrace, logicky seskupuje různé související senzory dohromady. Jako příklad lze uvést: **fbt** – sledovaní funkci jádra, **pid** – monitorování procesů uživatelského prostoru a **syscall** – sledování systémových voláni.
- Modul: Umístění skupiny senzorů. To může být název modulu jádra, ve kterém je senzor umístěn, nebo uživatelská knihovna. Například: **libsc.so** knihovna nebo modul jádra **ufs**.
- Funkce: Určuje funkci, ve které má tento senzor pracovat. To může být soukromá funkce v knihovně, například **printf()** nebo **strcpy()**.
- Jméno: To obvykle odráží účel senzoru. Například „entry“ nebo „return“ pro funkci nebo „start“ pro vstup-výstupní senzor. Pro sledování na úrovni příkazů je ukázán posun uvnitř funkce.

Výstup dat se provádí přes mezipaměť, odkud spotřebitel data převzal. Mezipaměť je nutná, aby uživatel mohl pracovat asynchronně bez blokování provozu jádra a aby se vyhnul velkému počtu přepínaní kontextu. Přijatá informace se zobrazuje na terminálu nebo v souboru. DTrace je schopna sledovat každou vrstvu softwaru, včetně zkoumání interakcí různých vrstev, tato interakce je zobrazena na obrázku 3.4.



Obrázek 3.4: Interakce DTrace s různými vrstvy softwaru [6].

DTrace je mocný a flexibilní prostředek pro monitorovaní ne jenom programů, ale i systému s jeho vnitřními voláními. Ale i přes to on se hodí ne pro všechny případy kde potřebujeme monitorovat aplikaci. Nevýhodou toho přístupu je to, že člověk potřebuje důkladně nastudovat dokumentací DTrace aby umět psát monitorovací scripty v jazyce **D**. Při tom možnosti sledovaní několika počítačů a sbíraní informaci o nich na jednom místě muže byt omezena.

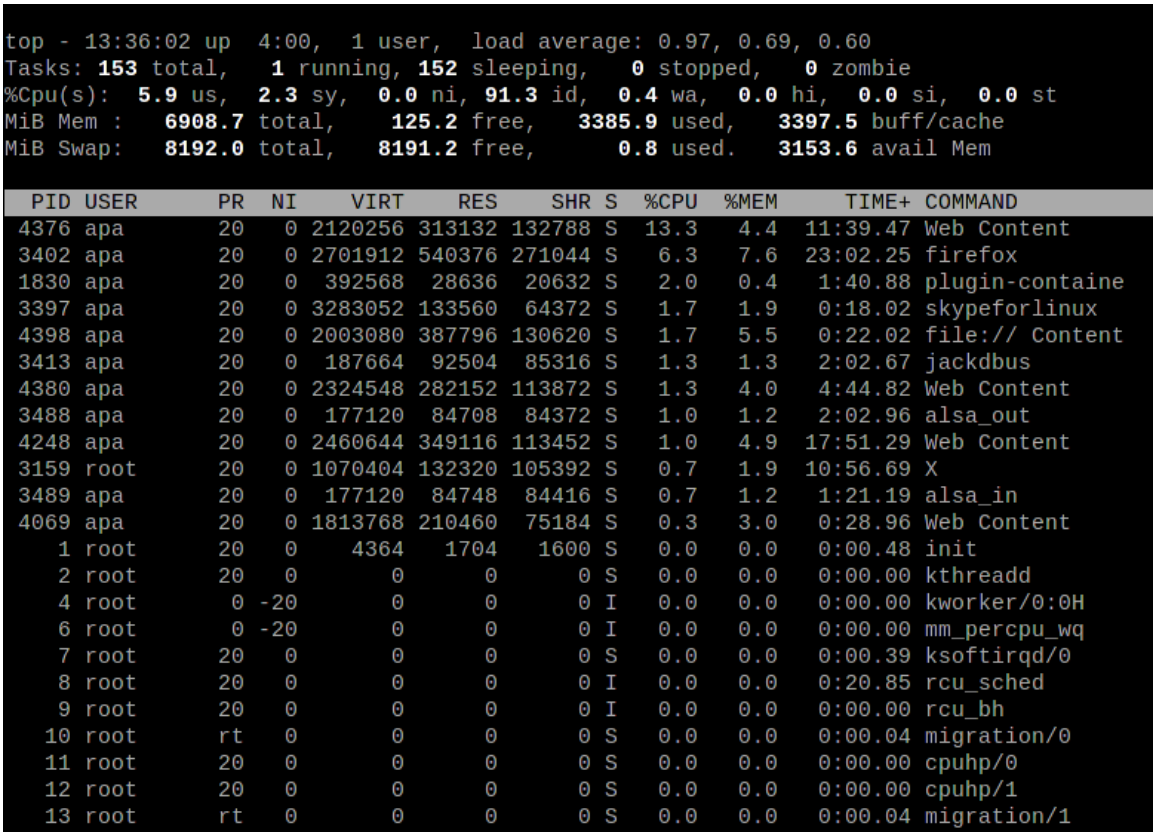
3.5 Monitorovací prostředky systému Linux

V operačním systému Linux existují některá vestavěná aplikace, která umožňují sledovat a zjišťovat skutečné příčiny problémů s výkonem. Také existují baličky monitorovacích programů, které nejsou vestavěné přímo do systému, ale jsou dostupné pro skoro každou distribuci Linux jako oficiální balík repositáře. V teto sekci je stručně popsaná charakteristika takových servisních programů.

top

Příkaz **top** je příkaz pro monitorování výkonu, který je často používán mnoha správci systému k monitorování výkonu Linux. Příkaz top se používá k zobrazení všech běžících a aktivních procesů v reálném čase v uspořádaném seznamu. Zobrazuje využití procesoru, využití paměti, *swap* paměť, velikost mezipaměti, velikost vyrovnávací paměti, ID procesu,

uživatele a další užitečné věci. Příklad výpisu při volaní **top** je na obrázku 3.5. **Top** je velmi vhodný příkaz pro monitorování a rozhodování v případě problémů.



Obrázek 3.5: Výpis příkazu **top**.

vmstat

Příkaz **vmstat** se používá k zobrazení statistik virtuální paměti, vláken jádra, disků, systémových procesů, I/O bloků, přerušení, aktivity procesoru atd. Příklad výstupu **vmstat** je na obrázku 3.6. Ve výchozím nastavení **VmStat** muže být nedostupný v systému Linux, v tom případě je třeba nainstalovat balíček **sysstat**, který obsahuje ten program.

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
0  0    776 368084 791160 2368228    0    0    60   192  438  213 15  3 80  2  0
```

Obrázek 3.6: Výsledek volaní **vmstat** je rozdělen do šesti sloupců - procs (procesy), memory (paměť), swap, io (I/O disku), system (systém/jádro), CPU(procesor).

Procs

r – počet procesů ve frontě pro zpracování procesorem (je-li hodnota > 0 - existuje zatížení procesoru)

b – počet procesů, které čekají na operace I/O (pokud je hodnota > 0 - na discích nebo souborovém systému je zatížení)

Memory

swpd – počet bloků přesunutých do swapu

free – volná paměť

buff – buffery paměti

cache – cache

Swap

si (swap in) – počet bloků za sekundu, které systém přečte ze sekce nebo swap souboru do paměti

so (swap out) – a naopak, počet bloků za sekundu, které systém přesune z paměti do swapu

IO

bi (blocks in) – počet bloků za sekundu načtených z disku

bo (blocks out) – počet bloků za sekundu zapsaných na disk

System

in (interrupts) – počet přerušení za sekundu

cs (context switches) – počet přepínání mezi úkoly

CPU

us (user time) – % času procesoru stráveného při provádění „uživatelsky definovaných" úkolů

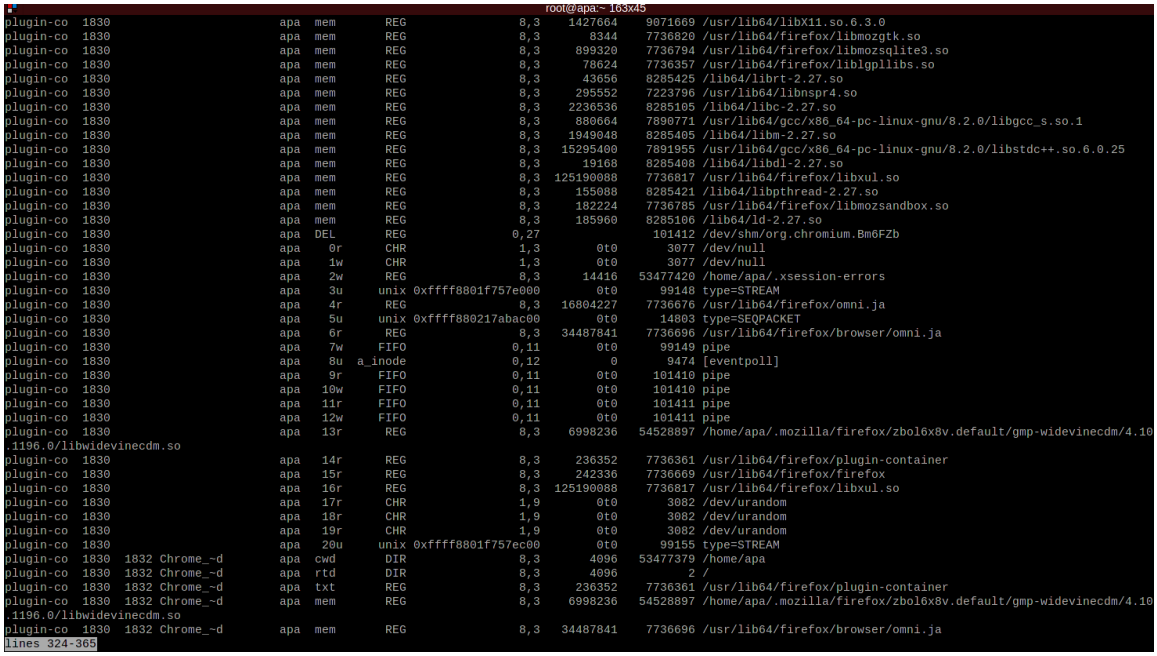
sy (system čas) – % času CPU, který se používá k provádění úloh jádra (síť, úlohy I/O, přerušení atd.)

id (idle) – % času v nečinnosti (čekání na úkoly)

wa – % času CPU, během čekání na operace I/O

Lsof

Příkaz **lsof** se používá v mnoha systémech Linux/Unix pro zobrazení seznamu všech otevřených souborů a procesů. Otevřené soubory zahrnují diskové soubory, síťové sokety, kanály, zařízení a procesy. Jedním z hlavních důvodů použití tohoto příkazu je, že disk nelze odpojit, kdy jsou použity nebo otevření některých souborů z něho. Pomocí tohoto příkazu lze snadno určit, které soubory se používají.



Obrázek 3.7: Výpis příkazu **lsof**.

NetStat

Užitečný příkaz a obslužný program, nazývaný **netstat**, umožňuje zobrazit informace o systémových připojeních pomocí protokolů UDP a TCP.

Program může fungovat tak, aby byl spuštěn každých n sekund a umožňuje přijímat následující informace ve formátu tabulky:

- Název protokolu (TCP nebo UDP)
- Localní adresa IP a číslo portu, které používá soketové připojení
- Vzdálená adresa IP (cílová adresa) a číslo portu používané připojením soketu
- Stav připojení (Listening), navázáno (Established) atd.

Velicé důležitou informaci je pravě stav připojení, ktery může nabivat nasledujíci hodnoty:

- CLOSE_WAIT – označuje pasivní fázi uzavření spojení, která začíná poté, co server obdrží od klienta zprávu FIN.
- CLOSED – spojení bylo přerušeno a zavřeno serverem.
- ESTABLISHED – klient navázal spojení se serverem po obdržení zprávy SYN ze serveru.
- FIN_WAIT_1 – klient inicioval uzavření spojení (odeslal zprávu FIN).
- FIN_WAIT_2 – klient přijal ze serveru zprávy ACK a FIN.
- LAST_ACK – server odeslal klientovi zprávu FIN.

- LISTEN – server je připraven přijmout příchozí spojení.
- SYN_RECEIVED – Server obdržel od klienta zprávu SYN a odeslal jí odpověď.
- TIMED_WAIT – klient poslal zprávu FIN na server a čeká odpověď na tuto zprávu.
- YN_SEND – zadané spojení je aktivní a otevřené.

Podle poskytnutých dat programem **netstat** a jejích analýzy lze zjistit jaký typ problému je v síťovém připojení a jak s nim pracovat dál.

ps

Příkaz **ps** je hlavním nástrojem, který správce systému používá ke sledování procesů. Verze tohoto příkazu se liší v argumentech a výstupním formátu, ale ve skutečnosti poskytují stejné informace. V zásadě je rozdíl ve verzích důsledkem různých způsobů vývoje systémů UNIX. Příklad jednoho z možných výpisu **ps** je na obrázku 3.8. Tento přikaz ma několik typů přepinaču [14]:

1. Přepínače UNIX, které mohou být seskupeny a musí jim předcházet pomlčka.
2. Přepínače BSD, které mohou být seskupeny a nesmí být používány s pomlčkou.
3. Dlouhé Přepínače GNU, kterým předchází dvě pomlčky.

Na obrázku dolů je zobrazena informace o běžících procesech. Tato informace rozdělena do sloupců, které mají následující význam [14]:

USER Jméno uživatele, který spustil proces

PID Identifikátor procesu

%CPU Procento času CPU přiděleného procesu

%MEM Část paměti (v procentech) používaná procesem

VSZ Virtuální velikost procesu

RSS Počet stránek paměti

TTY Identifikátor ovládacího terminálu

STAT Aktuální stav procesu:

R – běží
D – čeká na zápis na disk
S – neaktivní (< 20 s)
T – pozastaveno
Z – zombie
W – proces vyložen na disk
< – proces má vysokou prioritu
N – proces má nízkou prioritu

L – některé stránky jsou uzamčeny v paměti RAM

s – proces je vedoucím sezení

TIME Množství času procesoru stráveného na procesu

COMMAND Jméno a argumenty příkazu pro spuštění

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
apa       1830  0.8  0.4 392568 28644 ?        Sl   09:45   2:12 /usr/lib64/firefox/plugin-container /hom
apa       3352  0.0  0.0  13940  3092 ?        Ss   09:35   0:00 /bin/sh /etc/xdg/xfce4/xinitrc -- /etc/X
apa       3358  0.0  0.0  27064  2408 ?        S    09:35   0:00 /usr/bin/dbus-launch --exit-with-session
apa       3359  0.0  0.0  16092  2916 ?        Ss   09:35   0:00 /usr/bin/dbus-daemon --syslog --fork --p
apa       3365  0.0  0.2 338672 15328 ?        Sl   09:35   0:00 xfce4-session
apa       3369  0.0  0.0  44736  4792 ?        S    09:35   0:00 /usr/lib64/xfce4/xfconf/xfconfd
apa       3372  0.0  0.0  13192   328 ?        Ss   09:35   0:00 /usr/bin/ssh-agent -s
apa       3374  0.0  0.0 165920   532 ?        Ss   09:35   0:00 /usr/bin/gpg-agent --sh --daemon --write
apa       3376  0.4  0.3 199756 24728 ?        S    09:35   1:18 xfwm4
apa       3380  0.1  0.4 387396 29308 ?        Sl   09:35   0:22 xfce4-panel
apa       3382  0.0  0.4 670636 33040 ?        Sl   09:35   0:01 Thunar --daemon
apa       3384  0.0  0.4 463236 31392 ?        Sl   09:35   0:02 xfdesktop
apa       3385  0.0  0.2 346652 14416 ?        Ssl  09:35   0:01 xfsettingsd
apa       3395  0.2  3.0 1691056 215908 ?      SLl  09:35   0:41 telegram
apa       3397  0.1  1.8 3283052 133628 ?      Sl   09:35   0:19 /opt/skypeforlinux/skypeforlinux --execu
apa       3402  9.6  7.7 2701872 547672 ?      SLl  09:35  25:53 firefox
apa       3405  0.0  0.2 328172 19460 ?        Sl   09:35   0:00 /usr/libexec/polkit-gnome-authentication
apa       3408  0.0  0.0  61172  5020 ?        S    09:35   0:06 xscreensaver -no-splash
apa       3413  0.8  1.3 187664 92504 ?        SLsl 09:35   2:22 /usr/bin/jackdbus auto
apa       3428  0.0  0.1 563796 13356 ?        Ssl  09:35   0:00 xfce4-volumed
apa       3432  0.0  0.0 279200  6348 ?        Sl   09:35   0:00 /usr/libexec/gvfsd
apa       3488  0.8  1.1 177120 84708 ?        SLl  09:35   2:18 alsa_out -j xonar -d hw:DSX -c 4 -q 1
apa       3489  0.5  1.1 177120 84748 ?        SLl  09:35   1:31 alsa_in -j xonar-01 -d hw:DSX -q 1
apa       3494  0.0  0.0 274872  6856 ?        Sl   09:35   0:00 /usr/libexec/gvfs-udisks2-volume-monitor
apa       3516  0.0  0.4 412584 29220 ?        Sl   09:35   0:00 /usr/libexec/notification-daemon
apa       3523  0.0  0.2 596444 20448 ?        Sl   09:35   0:00 /usr/lib64/xfce4/panel/wrapper-1.0 /usr/
apa       3528  0.0  0.6 807756 47044 ?        Sl   09:35   0:03 /usr/lib64/xfce4/panel/wrapper-2.0 /usr/
apa       3538  0.0  0.5 411000 41468 ?        Sl   09:35   0:04 /usr/lib64/xfce4/panel/wrapper-2.0 /usr/
apa       3539  0.0  0.2 186724 14320 ?        S    09:35   0:00 /usr/lib64/xfce4/panel/wrapper-1.0 /usr/
apa       3540  0.0  0.2 189592 16688 ?        S    09:35   0:00 /usr/lib64/xfce4/panel/wrapper-1.0 /usr/
apa       3578  0.0  0.0 355308  6312 ?        Sl   09:35   0:00 /usr/libexec/gvfsd-trash --spawner :1.14
apa       3589  0.0  0.4 354032 31172 ?        S    09:35   0:00 /opt/skypeforlinux/skypeforlinux --type=
apa       3618  0.0  0.0 192124  5720 ?        Sl   09:35   0:00 /usr/libexec/gvfsd-metadata
apa       4069  0.3  4.0 1958372 284020 ?      Sl   09:35   0:55 /usr/lib64/firefox/firefox -contentproc
apa       4116  0.0  0.0  14204  3416 ?        S    09:36   0:00 /bin/bash /usr/share/netbeans-nb-8.2/pla
apa       4121  0.1  3.8 3507548 274324 ?      SNl  09:36   0:26 /opt/skypeforlinux/skypeforlinux --type=
apa       4248  6.7  4.9 2459244 353288 ?      Sl   09:36  18:08 /usr/lib64/firefox/firefox -contentproc
apa       4372 11.4 17.8 4769696 1265972 ?      Sl   09:36  30:37 /opt/oracle-jdk-bin-1.8.0.202/bin/java -
apa       4376  5.8  3.9 2113472 282972 ?      Sl   09:36  15:36 /usr/lib64/firefox/firefox -contentproc
apa       4380  1.8  4.0 2325572 286328 ?      Sl   09:36   5:05 /usr/lib64/firefox/firefox -contentproc
apa       4398  0.3  3.8 2040228 271776 ?      Sl   09:36   1:00 /usr/lib64/firefox/firefox -contentproc
apa      18494  1.4  2.9 3196624 208300 ?      Sl   13:16   0:42 /home/apa/.dropbox-dist/dropbox-lnx.x86_
apa      18582  0.3  2.1 3570776 153340 ?      Sl   13:17   0:09 texmaker
apa      20478  0.0  0.0 355180  6896 ?        Sl   13:36   0:00 /usr/libexec/gvfsd-network --spawner :1.
apa      20523  0.0  0.0 185532  4944 ?        Sl   13:36   0:00 /usr/libexec/dconf-service
apa      27190  0.6  0.8 702772 58232 ?        Sl   13:54   0:03 /usr/bin/python2.7 /usr/lib/python-exec/
apa      27484  0.0  0.1  19840  7636 pts/1    Ss+  13:54   0:00 /bin/bash
apa      27626  0.0  0.1  19840  7440 pts/2    Ss   13:54   0:00 /bin/bash
apa      28090  0.0  0.1  19844  7608 pts/0    Ss   13:54   0:00 /bin/bash
apa      29601  0.0  0.5 466532 36216 ?        Sl   13:59   0:00 ristretto /home/apa/Dropbox/bachelor_wor
apa      30092  0.0  0.0  18036  3720 pts/2    S+   14:01   0:00 man ps
apa      30105  0.0  0.0  15704  2912 pts/2    S+   14:01   0:00 /usr/bin/less
apa      31299  0.0  0.0  13292  2472 pts/0    R+   14:04   0:00 ps ux
```

Obrázek 3.8: Výpis příkazu **ps ux**. Výpis je ve formátu **BSD**, přepínač **x** zapíná vypisovaní jenom procesu současného uživatele a přepínač **u** formátuje výpis do uživatelsky orientovaného formátu.

tcpdump

Analyzátory síťového provozu se často používají k nalezení problémů v síti. **Tcpdump** je jedním ze zástupců této třídy programů, umožňuje poslouchat (zobrazovat/ukládat) a analyzovat síť na úrovni přenášených síťových paketů, rámců a dalších přenosových jednotek síťového provozu. V závislosti na konfiguraci sítě může **tcpdump** naslouchat nejen pakety určené pro danou MAC adresu, ale také broadcastové vysílání.

V závislosti na síťovém zařízení používaném pro připojení počítačů v síti existují následující způsoby sledovaní provozu:

- V síti založené na ethernetovém hubu je veškerý provoz z hubu dostupný pro všechny síťové hosty.
- V sítích založených na přepínačích je k dispozici hostiteli sítě pouze jeho provoz, stejně jako veškerý broadcastový provoz tohoto segmentu.
- Některé spravované přepínače mají funkci kopírování provozu z daného portu do monitorovacího portu.
- Při použití speciálních nástrojů (network tapů), které se připojuje přímo k počítačovému síťovému kabelu a vytváří kopii síťového provozu pro přenos do jiného zařízení.

TCPdump je těsně spojen s knihovnou **libpcap**. Tato knihovna byla vytvořena stejným týmem, který vyvinul **tcpdump**. Funkcí zachytávání paketů s nízkých úrovní, čtení zachycených paketů a jejích zapisování do souboru **tcpdump** bylo extrahováno a provedeno do knihovny [9]. S využitím **libcap** lze napsat svůj vlastní monitorovací program pro sledovaní síti. Také tuto knihovnu používají i jiné programy jako například **wireshark**, **WinPcap** atd.

3.6 Monitorovaní ve sféře řízení letového provozu

Na řídicích stanicích se monitorují různé věci jako síťová komunikace, její stabilita a kvalita. Také jde neustaly zápis přenášených zvukových dat mezi stanicemi a letadly, radarových dat, elektronických stripů atd. A pro sledováni softwaru běžícího přímo na stanici muže byt použity buď nějaký jednoduchý program třetích stran, který lze snadno rozjet a řídit, nebo jeho píšou samí vývojáři pro svůj software. A v obou případech ten software může neposkytovat dostatečnou sadu dat pro analýzu a připadnou rychlou reakci na chyby. Nebo samozřejmě monitorovací software je komerční a drahý. Proto významnou roli hraje software navrženy od začátku s ohledem na danou sféru využiti, a který je nekomerční produkt s otevřeným zdrojovým kódem.

Například společnost **EIZO** (https://www.eizoglobal.com/) používá za učelem monitorovaní systém **SafeGuard**. **SafeGuard** je řešení pro sběr dat a správu obsahu určené pro příjem více zdrojů z komunikačních rozhraní používaných v kritických prostředích, jako je řízení letového provozu. Jedná se o systém, který poskytuje vysoce bezpečný archiv podporující správu, analýzu a distribuci dat pro průmyslová odvětví, která usnadňují vyšetřování po incidentech, operací vyhledávání a záchrany, analýzu školení a výkonnosti, validaci a verifikaci.[15]

Ale monitorovaní stanic je jenom mála část velkého softwaru, hlavním předurčením kterého je snímání zvuku (analogové, digitální, VoIP), radarových dat (sériového nebo síťového), klávesnice a myši, videosignálů (analogové a digitální), signálů kamer a distribuovaných síťových dat. A dálnější centralizovaná správa získaných dat.

Dryhým přikladem monitorovacího systemu je **ICZ LETVIS TCM** od společnosti **ICZ LETVIS** (https://www.iczgroup.com/en/). **ICZ LETVIS TCM** je systém umožňující technické monitorování jednotlivých serverů a pracovních stanic z hlediska operačního systému. Kontroluje funkčnost, výkon a připojení jednotlivých serverů k systému **ICZ LETVIS**. Poskytuje také stručné informace o stavu sledované aplikace. Monitorovací systém umožňuje technický dohled nad stavem systému **ICZ LETVIS**. [7]

V tomto případě je monitorovací prostředek zvláštní aplikací, která je určena sledovaní přímo stanici a procesů. Ale tato aplikace navázaná na určitý software od určitého výrobce, tak že jí nelze použit s žádnou jinou aplikaci.

3.7 Shrnutí dosavadního stavu

Již od vzniku letového provozu a jeho hromadného využití, jeho nejdůležitějším aspektem byla bezpečnost jak lidí tak i samotných letadel. Pro splnění dostatečného úrovně bezpečnosti vznikla služba ŘLP (Řízení letového provozu). Vysokou spolehlivost a efektivitu teto služby zajišťuji jak špičkově odborníci tak i kvalitní software. Ale při běhu softwaru nikdy nelze spoléhat na to že on bude fungovat vždycky korektně a tak jak má. A proto existují různé monitorovací systémy, které pomáhají sledovat stav softwaru na řadicích stanicích. I při tom že existuje spousta různých systémů pro monitorovaní, každý z nich ma svoje nevýhody. Některé jsou komerční produkty, které nemají otevřený zdrojový kód a občas můžou byt dostatečné drahé. Jiné jsou jenom prostředky pro tvorbu monitorovacích skriptů nebo programů.

Kapitola 4

Použité prostředky

V teto kapitole jsou popsané různé prostředky použité při vývoje nového systému monitorovaní běžících aplikaci, důvody jejích použití, jaké mají výhody a jejích vlastnosti obecně.

4.1 Operační systém

Operační systém je systémový software, který spojuje aktivuje všechny technické komponenty jakéhokoliv počítače. Operační systém funguje jako prostředník mezi uživatelem a počítačovým hardwarem. Účelem operačního systému je poskytnout prostředí, ve kterém může uživatel provádět programy pohodlným a efektivním způsobem. Operační systém je software, který spravuje počítačový hardware. Hardware musí poskytovat vhodné mechanismy pro zajištění správného provozu počítačového systému a pro zabránění uživatelským programům rušení korektního fungování systému.[16]

Operačním systémem pro implementací monitorovaní byl vybraný **Linux**. Důvodem pro takový výběr je to, že tento systém je často použity ve sféře řízení letového provozu jak na řidičích stanicích tak i na serverech. Hlavními příčiny pro to jsou jeho otevřený zdrojový kód, bezpečnost, malá velikost samotného systému a rychlost. Také důležitou příčinou je jeho jádro, které mezi různá distribuce zůstává velice podobné, a proto lze snadno implementovat software, který by byl funkční mezi různými distribucemi.

Jádro operačního systému

Těžko přesně říct co je jádro operačního systému a co je samotný operační systém. Operační systém lze charakterizovat jako „rozšiření“ jádra o nějakou funkcionalitu, nejčastěji o nějaké programy, který umožňuje uživateli přístup k hardwarovým prostředkům. Jádro je hlavní součástí každého operačního systému. To je také program napsaný v nějakém programovacím jazyce a zkompilovaný do spustitelného souboru. Na rozdíl od jiných programů je však jádro vždy načteno jako první a pak neustále „sedí“ v určité oblasti RAM. To znamená, že se jedná o program, který je vždy v aktivovaném stavu a spolupracuje na jedné straně s hardwarem a na druhé straně se systémovými a uživatelskými programy.

Distribuce

Distribuce je sbírka programů (balíčků), sady nástrojů sestavených dohromady na jádru systému. Linuxové jádro je vydáváno centrálně a distribuce Linuxu, s určitým množstvím znalostí a dovedností, může být shromažďována kýmkoli od nuly nebo na základě existující

distribuce. V současné době existuje velké množství distribucí operačního systému Linux. Z teto sady existuje řada systémů, které jsou vytvořeny na komerčním základě, avšak většina distribucí je volně distribuována podle podmínek licence GPL[1]. Uživatel tak má možnost si vybrat některou z distribucí, kterou má rád, nebo, jak již bylo zmíněno, se znalostmi a dovednostmi, může sestavit svou vlastní podle přání.

Nejpopulárnější distribucí jsou:

- Ubuntu – Jeden z nejpopulárnějších operačních systémů Linuxu. Vyvinutý společností **Canonical** na základě **Debianu**. Verze dlouhodobé podpory jsou velmi stabilní a neustále dostávají aktualizace.
- Linux Mint – založený na Ubuntu a vyvinutý nezávislým vývojovým týmem. Systém má většinu výhod Ubuntu - ve formě velkého počtu balíčků a úložišť PPA. Také je pro něj vhodná většina instrukcí z Ubuntu a systém je poměrně stabilní.
- CentOS – vyvinutý komunitou na základě komerční distribuce Red Hat Enterprise Linuxu. Jeden z nejlepších systému pro použiti na serverech. Pro mnoho serverových programů, speciálně vytvořené repositáře od vývojářů, podobné PPA pro Ubuntu.
- Gentoo – distribuce, která pomáhá dobře porozumět tomu, jak se instaluje Linux, jaké procesy se během toho vyskytují, jaký software se vytváří a tak dále.
- Fedora – navržen jako bezpečný, univerzální operační systém. Je vyvinut v šestiměsíčním cyklu vydání pod záštitou projektu Fedora.

Monitorovací program byl implementován v repositáři Ubuntu s využitím systémových knihoven teto distribuci. Ale testovaní proběhlo také v repositářích Gentoo a CentOS, aby ověřit přenositelnost a funkčností na jiných systémech.

4.2 Jazyk C++

C++ je univerzální programovací jazyk. S výjimkou drobných detailů je **C++** nadmnožinou programovacího jazyka **C**. Kromě funkcí **C**, **C++** poskytuje flexibilní a efektivní mechanizmy pro definování nových typů. Programátor může rozdělit aplikaci do spravovatelných částí definováním nových typů, které přesně odpovídají koncepcím aplikace. Tato technika pro konstrukci programu se často nazývá *abstrakce dat*. Objekty některých uživatelem definovaných typů obsahují informace o typu a metody. Tyto objekty mohou být používány pohodlně a bezpečně v kontextech, ve kterých jejich typ nemůže být určen při kompilaci. Programy používající objekty takových typů se často nazývají objektové. Pokud jsou tyto techniky používány dobře, mají za následek kratší, srozumitelnější a snadnější údržbu programů. Klíčovým konceptem v jazyce C ++ je třída. To je uživatelem definovaný typ, který poskytují skrývání dat, zaručenou inicializaci dat, konverzi implicitního typu pro uživatelem definované typy, dynamické typování, správu uživatelsky řízené paměti a mechanismy pro přetížení operátorů.[17]

Programovací jazyk **C++** je široce používán pro vývoj softwaru. Vytváření různých aplikačních programů, ovladačů zařízení, vývoj operačních systémů, stejně jako videoher a mnoho dalšího. Dal bude popsaná krátká historie jazyka a jeho standardů, některé jeho osobitosti a výhody, které on přináší.

[1] GPL – General Public License, více viz https://www.gnu.org/licenses/gpl-3.0.en.html

Historie jazyka a jeho standardů

Jazyk vznikl na počátku 80. let, kdy **Bjarne Stroustrup**, zaměstnanec společnosti Bell Labs, přišel s řadou vylepšení jazyka C pro své vlastní potřeby. On se rozhodl doplnit jazyk **C** (nástupce BCPL) o funkce dostupné v jazyce **Simula** [17]. Stroustrup přidal schopnost pracovat s třídami a objekty. V důsledku toho se ukázalo, že praktické modelovací problémy lze efektivně řešit jak z hlediska doby vývoje (vzhledem k použití tříd **Simula**), tak z hlediska času výpočtu (v důsledku rychlosti **C**). Nejprve byly do **C** přidány třídy (s enkapsulací), dědičnost tříd, kontrola typů, inline funkce a výchozí argumenty. Nový jazyk, neočekávaně pro autora, se stal velmi oblíbeným mezi kolegy a brzy ho Stroustrup nemohl jeho podporovat sám.

Jak **C++** se vyvíjel, jiné nástroje byly přidaný do něho, které překrývali schopnosti **C** konstrukcí, a proto několikrát vznikala otázka odmítnutí jazykové kompatibility, aby odstranit zastaralé konstrukty. Ale kompatibilita však byla zachována.

V roce 1983 byly do jazyka přidány nové funkce, jako jsou virtuální funkce, přetížení funkci a operátorů, odkazy, konstanty, uživatelská kontrola nad správou volné paměti, vylepšená kontrola typu a nový styl komentářů. Výsledný jazyk přestal být jednoduše rozšířenou verzí klasického C a byl přejmenován z „C with classes“ na „C++“. Jeho první komerční vydání proběhlo v říjnu 1985. Před zahájením oficiální normalizace **C++** byl jazykem, který vyvinul především Stroustrupem sam jako reakce na požadavky programátorské společnosti. Funkce standardních jazykových popisů vykonávali publikace autora jazyka.[18]

První standard jazyka **C++98** byl nakonec schválen až v roce 1998. V roce 2003 byl vydán standard **C++03**, což je zdokonalení standardu **C++98**.Nejvýznamnější změny v jazyce nastaly v normě **C++11**, jejíž vývoj byl dokončen v roce . V roce 2014 byl vydán standard **C++14**, který neobsahoval významné změny, ale pouze eliminoval řadu vad předchozího standardu. V roce 2017 byl vydań poslední pro dnešni dobu standard jazyka **C++17**. **C++20** neoficiální název normy pro programovací jazyk **C++**, který se očekává po současném standardu.

Osobitosti jazyka

C++ STL (Standard Template Library) je výkonná sada šablonových tříd pro poskytování univerzálních tříd a funkcí, které implementují mnoho populárních a běžně používaných algoritmů a datových struktur, jako jsou vektory, seznamy, fronty a haldy. Jádrem **C++ STL** jsou tři dobře strukturované komponenty – kontejnery, algoritmy a iteratory. Kontejnery slouží ke správě sbírek objektů určitého druhu. Existuje několik různých typů kontejnerů, jako je deque, list, vector, map atd. Algoritmy působí na kontejnery. Oni poskytují prostředky, kterými budete provádět inicializaci, třídění, vyhledávání a změnu obsahu kontejnerů. Iterátory se používají k procházení prvky sbírek objektů. Tyto sbírky mohou být kontejnery nebo podmnožiny kontejnerů. [4]

Hlavními výhodami jazyku **C++** jsou:

- V něm jsou podporovány různé styly a technologie programování, včetně OOP, zobecněného programování a metaprogramování (šablony, makra).
- Předvídatelné provádění programu je důležitou výhodou pro budování systémů v reálném čase. Veškerý kód implicitně generovaný kompilátorem pro implementaci jazykových funkcí (například při převodu proměnné na jiný typ) je definován ve standardu. Také striktně definovaná místa programu, ve kterých je tento kód prováděn. To umožňuje měřit nebo vypočítávat dobu odezvy programu na externí událost.

- Automatické volání destruktorů objektů, když jsou zničeny, a to v pořadí opačném než volání konstruktorů. To zjednodušuje (stačí jenom deklarovat proměnnou) a umožňuje spolehlivější uvolnění prostředků (paměť, soubory, semafory atd.), a také umožňuje provádět přechody mezi stavy programu, které nejsou nutně spojeny s uvolňováním prostředků (například protokolování).
- Operátory-funkce definované uživatelem umožňují stručně zapisovat výrazy přes uživatelem definované typy v přirozeném algebraickém tvaru.
- Jazyk podporuje koncepty fyzické (const) a logické (mutable) konstanty. Tím je program spolehlivější, protože umožňuje kompilátoru například diagnostikovat chybné pokusy o změnu hodnoty proměnné. Deklarace stálosti dává programátorovi, který čte text programu, další myšlenku správného používání tříd a funkcí, a může být také tipem pro optimalizaci. Přetížení funkcí-členů na základě konstantnosti umožňuje definovat cíle volání metody zevnitř objektu (konstantní pro čtení, nekonstantní pro změnu). Deklarace *mutable* umožňuje udržovat logickou stálost při použití vyrovnávacích pamětí a odloženého vyhodnocování.
- Pomocí šablon je možné vytvářet generické kontejnery a algoritmy pro různé typy dat, stejně jako specializovat a vypočítávat ve fázi kompilace.
- Schopnost simulovat jazykové rozšíření pro podporu paradigmat, které nejsou přímo podporovány kompilátory. Například **std::bind** umožňuje vázat argumenty funkcí.
- Multiplatformní software. Standard jazyku má minimální požadavky k počítači pro spouštění skompilovaných programů. Pro určení vlastností systému na kterém poběží aplikace je ve standardní knihovně přítomna odpovídající funkcionalita (například std::numeric_limits<T>). Kompilátory jsou k dispozici pro velké množství platforem, v **C++** se vyvíjejí programy pro širokou škálu systémů.
- Efektivita. Jazyk je navržen tak, aby programátor měl maximální kontrolu nad všemi aspekty struktury a pořadí provádění programu. Žádná z jazykových funkcí, která vede k další režii, není povinná – pokud je to nutné, jazyk vám umožní zajistit maximální efektivitu programu.
- Je možnost pracovat na nízké úrovni s paměti.
- Vysoká kompatibilita s jazykem **C**, která umožňuje používat veškerý existující kód **C** (kód v tomto jazyce lze zkompilovat s kompilátorem **C++** s minimálními úpravami). Knihovny napsané v jazyce **C** lze obvykle zavolat přímo z **C++** bez jakýchkoli dalších nákladů, včetně funkcí zpětného volání, které umožňuje knihovnám napsaným v jazyce **C** volat kód napsaný v **C++**.

Hlavními důvody pro výběr jazyka **C++** byli jeho rychlost, možnost pracovat na nízké úrovně, objektová orientovanost a možnost programovat ve vice vláknech. Také na základě všech vlastnosti jazyka lze udělat závěr, že tento jazyk je efektivním a dostatečně flexibilním prostředkem pro řešení takového úkolu jako monitorovaní aplikaci a systému.

4.3 Knihovny

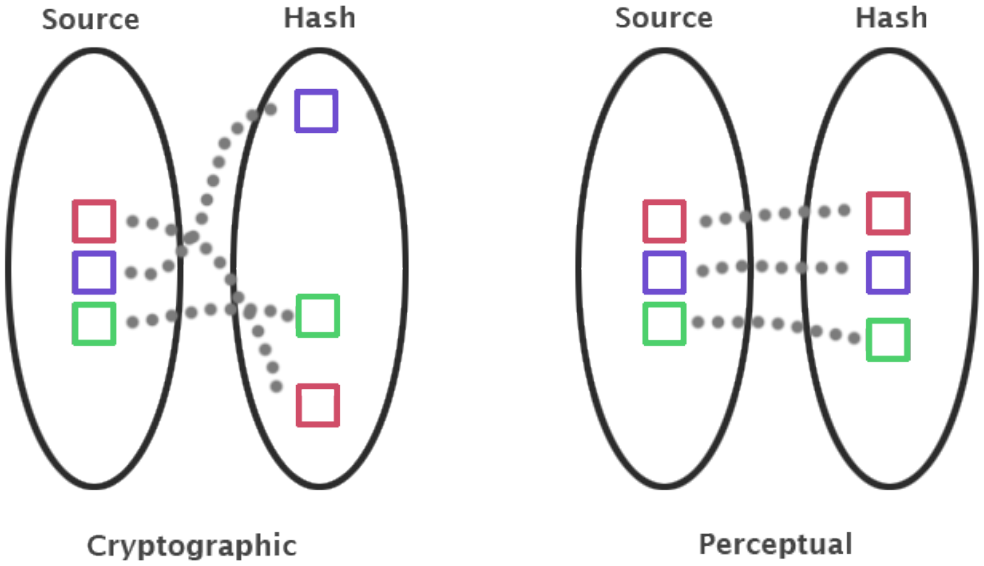
V teto kapitole popsané knihovny, které byli použity při implementaci aplikaci. Jejich funkcionalita a vyhody.

pHash[2]

Percepční hash (perceptual hash) je otisk multimediálního souboru odvozeného z různých vlastností z jeho obsahu. Na rozdíl od kryptografických hašovacích funkcí, které se spoléhají na lavinový efekt malých změn na vstupu, což vede k drastickým změnám ve výstupu, jsou percepční hašování navzájem "blízké", pokud jsou vlastnosti podobné. **pHash** je open source softwarová knihovna, která implementuje několik percepčních hashovacích algoritmů a poskytuje C-like API pro použití těchto funkcí ve vašich vlastních programech. Knihovna **pHash** je napsána v C++.[11]

Pomocí **pHash** lze najít nejen zmenšené obrázky, ale také obrazy, na kterých byly aplikovány nápisy, vodoznaky, části, kde byla provedena korekce barev nebo řez. Pro monitorovací aplikaci důležitou části teto knihovny byli hašovací a porovnávací funkce, pomoci kterých lze zjistit procent o jaky se změnil obrázek.

Percepční hash ma velký rozdíl s kryptografickým hashem. A to takový, že kryptografické hašovací funkce mohou být použity k mapování citlivých informací do hašovacích hodnot s vysokou disperzí, což za důsledek má to, že i menší změny ve zdrojových informacích, produkují významně odlišné hašovací výsledky. Proto z porovnaní dvou kryptografických hashů obvykle lze určit, jestli oni pocházejí ze stejného zdroje. Ale nelze však měřit podobnost dvou kryptografických hashů, aby zjistit podobnost zdrojů. Percepční hashe jsou další kategorií hashovacích funkcí, které mapují zdrojová data do hashů při zachování korelace. Tyto typy funkcí umožňují provádět smysluplná srovnání hashů za účelem nepřímého měření podobnosti mezi zdrojovými daty. [10]



Obrázek 4.1: Rozdíl kryptografického a percepčního hashy.[10]

Tato knihovna pro počítaní hashu obrázku vyžaduje využiti jiné knihovny pro práci s obrázky, a to je knihovna **CImg**.

[2] https://www.phash.org/ – knihovna pro porovnaní percepčního hashe pHash

CImg[3]

CImg je open-source C++ knihovna nebo sada nástrojů pro zpracování obrazu. Jedná se především o (velký) jednoduchý hlavičkový soubor CImg.h poskytující sadu C++ tříd a funkcí, které lze použít ve vlastních zdrojích, načíst/uložit, spravovat/zpracovávat a zobrazovat obecné obrázky. Je to vlastně velmi jednoduchá a příjemná sada nástrojů pro práci s obrázky v C++: Stačí začlenit hlavičkový soubor CImg.h, a už lze zpracovávat obrázky v programech C++.[3]

Knihovna **CImg** má následující strukturu:

- Všechny třídy a funkce knihovny jsou definovány v jmenném prostoru *cimg_library*, který obsahuje všechny funkce knihovny a zabraňuje kolizím, ke kterým může dojít při přidávání jiných hlavičkových souborů do projektu. Obvykle se tento jmenný prostor použije jako standardní volání:

```
#include "CImg.h"
using namespace cimg_library;
```

- Jmenný prostor `cimg_library::cimg` definuje sadu nízkoúrovňových funkcí a proměnných používaných v knihovně.
- Třída `cimg_library::CImg<T>` je hlavní třída knihovny, jejíž instance reprezentují entitu (obraz) až 4-dimenzionální (v rozsahu od jednorozměrných skalárních až po trojrozměrné sady pixelů), s šablonovými typy pixelů.
- Třída `cimg_library::CImgList<T>` představuje seznam obrázků `cimg_library::CImg`. Lze jej použít například k uložení sekvence snímků, rámců atd.
- Třída `cimg_library::CImgDisplay` zobrazuje obrázky nebo sady obrázků v grafickém prostředí. Lze říci, že kód této třídy je velmi závislý na systému, ale ve skutečnosti to nevadí programátorovi, protože proměnné prostředí jsou automaticky nastaveny knihovnou **CImg**.
- Třída `cimg_library::CImgException` (a její podtřídy) používá knihovna ke zpracování výjimek při výskytu chyb. Výjimky jsou zpracovávány pomocí `try {..} catch (CImgExceptio` Podtřídy umožňují přesně určit typ chyby.

Znalost těchto čtyř tříd a jmenného prostoru stačí k plnému využití funkčnosti knihovny **CImg**.

Libprocps

Librocps je nejdůležitější knihovna pro monitorovaní, která poskytuje rozhraní pro načítaní informace o systému a jeho procesech. Tato knihovna je součásti balíčku **procps-ng**, často se ho nazývají **procps**. Instalované programy z balíčku jsou: free, kill, pgrep, pkill, pmap, ps, pwdx, skill, slabtop, snice, sysctl, tload, top, uptime, vmstat, w, watch. A instalovaná knihovna je libprocps.so. Tento balíček se používá ve mnoha Linuxových distribucích. **Procps** načte informace o procesu z adresáře **/proc**.

[3] https://sourceforge.net/projects/cimg/ – knihovna pro zpracovaní obrazu CImg

Souborový systém **/proc** je mechanismus pro jádro a jeho moduly, který umožňuje posílat informace do procesů (odtud název /proc). Pomocí tohoto virtuálního souborového systému lze pracovat s vnitřními strukturami jádra, získávat užitečné informace o procesech a měnit nastavení (změnou parametrů jádra) za běhu. Systém souborů **/proc** se nachází v paměti, na rozdíl od jiných systémů souborů, které se nacházejí na disku.

Dal jsou některé důležité soubory z adresáře **/proc**:

- **proc/cpuinfo** – informace o procesoru např. model, značka, velikost mezipaměti atd.
- **proc/meminfo** – informace o RAM, velikosti swapu atd.
- **proc/mounts** – seznam připojených souborových systémů.
- **proc/devices** – seznam zařízení.
- **proc/filesystems** – podporované souborové systémy.
- **proc/modules** – seznam načtitelných modulů.
- **proc/version** – verze jádra.
- **proc/cmdline** – seznam parametrů předaných jádru při startu.

Souborový systém **/proc** je také zdrojem informací o běžících procesech. V adresáři **/proc** lze uvidět jiné adresáře, jejichž jména se skládají z čísel – jedná se o procesní informace. Název adresáře odráží identifikátor procesu (PID). Uvnitř těchto adresářů jsou soubory obsahující důležité informace o procesech – stavu, prostředí atd. Jednoduchý přístup pravě k teto informaci a informaci o systému poskytuje knihovna **procps**.

Původní autor měl málo času na procps. V roce 1997, Albert Cahalan napsal nový program **ps** pro balíček. V roce 2001 se Rik van Riel si vybral starý kód v Red Hat CVS (Concurrent Versions System) a začal přidávat opravy. Mezitím, ostatní lidé zlepšili procps mnoha způsoby. V roce 2002, Albert přesunul procps na `http://procps.sourceforge.net/`. Toto bylo děláno aby zajistil, že roky testování a opravy chyb nebudou ztraceny. Hlavní číslo verze bylo změněno na 3, částečně proto, aby se předešlo matoucím uživatelům a částečně proto, že byl přepracován top program. Poté, co se vývoj v podstatě zastavil na *sourceforge.net*, našel projekt v roce 2011 nový domov na adrese *gitorious.org/procps*. To představuje fork procps pro Debian, Fedora a openSUSE. Aby se předešlo nejasnostem a potenciálním střetům jmen, je balíček nyní známý jako **procps-ng** (příští generace), číslo verze bylo změněno na 3.3.0 a „so-name“ knihovny na **libprocps.so**. V roce 2015 Gitorious koupil Gitlab a projekt se stěhoval do nového domova na `https://gitlab.com/procps-ng`.[12]

Cereal[4]

Serializace je uložení/transformace objektu nebo stromu objektů do libovolného formátu, takže později mohou být objekty z tohoto formátu rekonstruovaný. Proces rekonstrukce objektu z určitého formátu zpátky se nazývá deserializace. Serializace se používá například k uložení stavu programu (tj. některých jeho objektů) mezi starty. Nebo pro přenos dat mezi různými instancemi programu (nebo různých programů), například prostřednictvím sítě.

[4] `http://uscilab.github.io/cereal/index.html` – serializační knihovna cereal

Hlavní myšlenkou je, že serializovaný formát je množina bajtů nebo řetězec, který lze snadno uložit na disk nebo přenést, na rozdíl od samotného objektu. To znamená, že úkol uložení objektu/skupiny objektů je omezen na jednoduchý úkol uložení sady bajtů nebo řetězce.

Cereal je serializační knihovna C++11, která se skládá pouze z hlavičkových souborů. Ona bere libovolné datové typy a reverzibilně je převádí do různých reprezentací, jako jsou kompaktní binární kódování, XML nebo JSON. Knihovna byla navržená tak, aby byla rychlá, lehká a snadno se rozšiřovala, také nemá žádné vnější závislosti a lze je snadno spojit s jiným kódem nebo použít samostatně. Podpora serializace pro téměř každý typ ve standardní knihovně C++ vychází je přimo z krabice. **Cereal** také plně podporuje dědičnost a polymorfismus. Vzhledem k tomu, že byla napsáná jako minimální, rychlá knihovna, neplní stejnou úroveň sledování objektů jako ostatní serializační knihovny, například boost. V důsledku tohoto ukazatele a odkazy nejsou podporovány, nicméně inteligentní ukazatele (jako `std::shared_ptr` a `std::unique_ptr`) není problém.[2]

Také tato knihovna je podobná `boost::serialization` a kompatibilní s ní, proto při použiti teto knihovny v projektu kde už je použity boost neměli by vzniknout problémy.

Kapitola 5

Návrh

V teto kapitole jsou popsaná situace zamrznutí jednotlivých komponent systému, shrnut současný stav monitorovacích prostředků, navrženo ce by se mělo byt změněno a vylepšeno. Také v kapitole je popsaná struktura nové monitorovací aplikaci, vlastnosti, které ona má mít, resp. jakosti klienta, serveru a jejích komunikaci.

5.1 Situace zamrznutí komponent systému

Ve sféře řízení letového provozu je velice důležitě aby software pracoval bezpečně, bez chyb a výpadků. Ale občas v aplikacích můžou nastat chyby, které například nebyli odhalené během jeho testovaní. Tak v praxi nejčastější chyby, které můžou vyvolat padnutí aplikaci nebo zamrznuti obrazovky jsou:

- Přeplnění systémového disku (např. logami nebo výstupem aplikaci, jestli ona vypisuje něco do souboru atd.)
- Vyčerpání pamětí RAM
- Nadměrné využití procesoru (např. v situaci, kdy jedno z vláken aplikaci běží v nekonečné smyčce bez žádného čekaní)
- Interní chyba aplikaci
- Selhání databázového serveru

A z důvodu toho, že tyto situace se vyskytuji nejčastěji, vyřešil jsem, že monitorovací aplikace bude sledovat využitou procesem operační paměť a CPU. Také aplikace měla by kontrolovat místo v rootovském adresáři na stanici, protože občas pravě nedostatek místa na disku může negativně ovlivnit funkčnost stanici. Pro zjištění interních chyb, které ovlivní GUI, monitorovaná bude obrazovka stanice a změny, které na ni probíhají. Co se tyká databázového serveru, obvykle pro jeho sledovaní se používají jiné prostředky a proto monitorovací klient nebude kontrolovat jeho běh.

5.2 Požadavky na monitorovací aplikaci

V dnešní době existuje velké množství monitorovacích aplikaci jak komerčních tak i s otevřeným zdrojovým kódem. Některé z nich jsou obecného charakteru a sleduji například

běh několika serverů nebo obecně systému, což občas nesplňuje požadavky na monitorovací systém pro sféru ŘLP. Co se tyká aplikaci popsaných v kapitole 3, každá z nich má svoje výhody a nevýhody. Některé z nich existují přímo pro monitorovaní aplikaci letového provozu, ale cena a uzavřenost kódu je problémem pro některé firmy. Jiné aplikaci vyžaduji znalost zdrojového kódu monitorované aplikaci a jeho přímou změnu. Také některé z nástrojů i přes to že poskytují velké množství užitečné informaci, ale nejčastěji každá aplikace analyzuje jenom část toho co je zapotřebí, a navíc muže byt problematické nastavit jejích komunikaci z nějakým kontrolním serverem.

Z toho plyne, že hlavními požadavky na novou monitorovací aplikaci, vzhledem k zanalyzovaným prostředkům, budou otevřeny zdrojový kód, rychlost, jednoduchost použití, spolehlivost a výkonnost. Také důležitým požadavkem na sledovací systém je jeho přenositelnost mezi různé distribuce Linux. Z toho důvodu součásti kódu samotné aplikaci budou i zdrojové kódy použitých knihoven, aby se dalo nakonfigurovat a zkompilovat je přímo na použitém operačním systému.

5.3 Struktura aplikaci

Ve sféře řízení letového provozu systémy mají následující součástky:

- Pracovní stanici, za kterými sedí řidiče a kontroluji lety, provívají komunikace mezi sebou a letadly.
- Server, který se používá pro komunikaci stanic a nahrávaní veškerých jednaní.
- Databázový server, který zpravidla běží na jiném počítači něž komunikační server, a může se nacházet na úplně odlišném stanovisku, než kontrolní server a stanici. On slouží pro inicializaci stanic a kontrolu průběhu cvičení, kdy stanici jsou v simulačním režimu.

Z důvodu takové struktury systému letectví, vyřešil jsem, že monitorovací aplikace bude mít podobnou strukturu klientu a serveru, bez použití databázového serveru. Aplikace bude se skládat ze serveru, který bude zodpovědný za sběr, analýzu a zobrazení informaci od klientů. A sami monitorovací klienty, které budou běžet na stanicích, sbírat užitečná data, a odesílat na server. Server a klient budou mít některé společné části například jako komunikační rozhraní, některá struktury pro uchovávaní dat, sterilizační knihovna atd. Ale většina funkcionality bude odlišná.

5.4 Klient-Server architektura

5.4.1 Klient

Klient bude představovat zvláštní aplikaci, která při spuštění bude vyžadovat identifikační číslo procesu pro sledovaní a udaje pro komunikaci jako multicastová adresa a port, kde bude poslouchat server a kam budou posílat data klienty. V případě, že ID procesu nebude zadáno, klient bude sledovat změny na obrazovce a obsazené místo. Ne jednom počítači bude možné spustit vice klientů, například jeden pro monitorovaní stanici a několik pro sledovaní procesů. Sbíraná data budou ne jenom o konkretním procesu ale také o systému. Nejdůležitější data pro sledovaní jsou: využita operační paměť aplikaci, procesorový čas, doba běhu procesu, množství obsazeného místa na disku. Navíc klient bude umět detekovat

zamrznutí aplikaci pomoci následujícího algoritmu. Jednou za určitý časový interval klient bude dělat screenshot obrazovky a porovnávat výsledný obrázek s předchozím screenshotem pomoci knihovny pHash 4.3. Porovnávají se ne sami obrázky a jejích percepční hashy, tento způsob je rychlejší a méně nákladný než porovnávání každého pixelu. Ve výsledku lze dozvědět jak se změnil obrázek a poslat tuto informaci na server.

5.4.2 Server

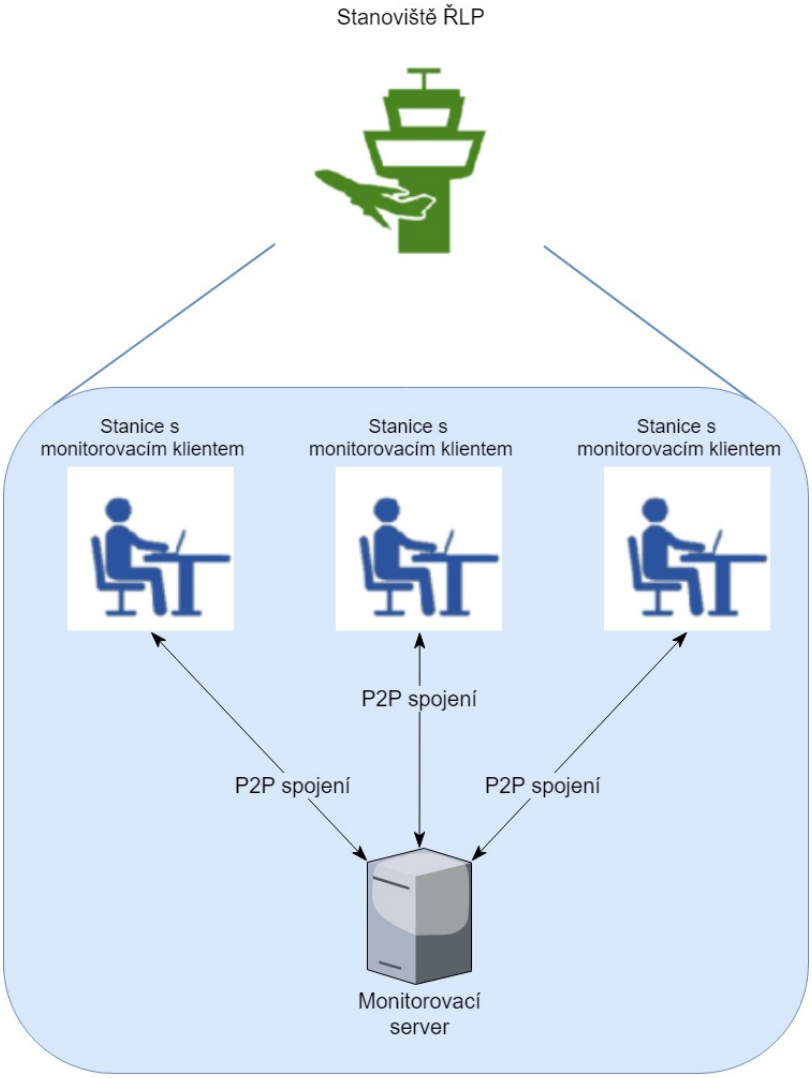
Server stejně jako klient bude představovat zvláštní spustitelnou aplikaci, která bude požadovat zadaní komunikačních parametrů při startu. Hlavní funkci serveru jsou sbíraní a zobrazení sledovaných údajů získaných pomoci komunikaci s klientem. Kromě údajů o procesu a systému, které posila klient, server bude evidovat čas, kdy přišla poslední zprava ze stanici a čas kdy přišla zprava s konkretní informaci. Také server ma konfigurační soubor, kde lze nastavit hraniční hodnoty pro konkretní sledovanou položku, například minimální a maximální využití procesoru, nebo maximální obsazené místo na disku atd. Při překročení těchto hodnot, server označí položku červenou barvou a vypíše varovaní do speciálního pole.

5.4.3 Komunikace

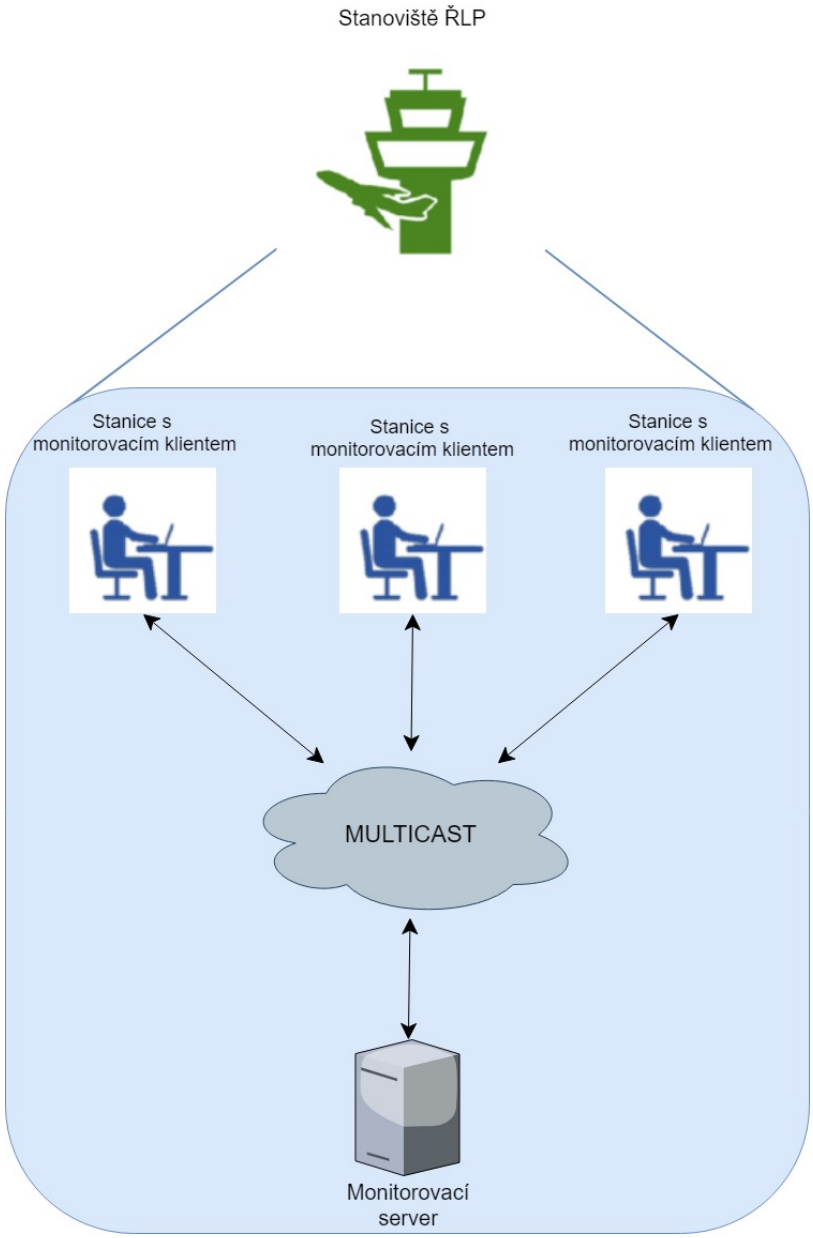
Na začátku jsem měl dvě varianty komunikaci mezi serverem a klienty. První byl takový, že si každý klient vytvoří Peer-to-peer spojení se serverem, a budou si posílat data každý přes svoje spojení. Příklad takového systému je na obrázku 5.1. Druhá varianta byla, že každý klient se připojí na jedinou multicastovou adresu, stejně jako server a tímto způsobem server bude poslouchat veškeré zprávy od klientů a klienti posílat sledovanou informaci do stejné multicastové skupiny. Při tom server bude rozlišovat stanice podle IP odesilatelé paketu a monitorovací proces podle obsahu samotného paketu s daty. Příklad systému používajícího multicast je na obrázku 5.2.

Komunikace bude probíhar pomoci zprav. Každá zprava bude mít svuj typ a vyznam. Na začatku napadli mě několik typů zprav, a to jsou:

1. INFO – zprava s daty o konkretním procesu od konkretní stanici.
2. STATION_INFO – zprava s daty o stanici, a konkretně o obsazeném prostoru v rootovské složce a o změnách obrazovky na monitoru.
3. REGISTER – zprava použita pro registraci klientu k serveru. Slouží k tomu, aby server věděl, které stanici jsou monitorovaný.
4. KEEP_ALIVE – zprava od klienta k serveru, která slouží pro signalizaci o tom že klient stále žije. Tato varianta signalizací je preferovaná, protože vyžaduje jenom jednu zprávu místo dvou.
5. PING a REPLY – druhá varianta signalizačních zprav, kde PING posílá server ke klientovi, o kterém chce zjistit jestli žije. REPLY zprava s odpovědí od klienta na přijatou zprávu PING, která bude signalizovat o správném běhu monitorovacího klienta.



Obrázek 5.1: Příklad Peer-to-peer komunikaci



Obrázek 5.2: Přiklad multicastové komunikaci.

Kapitola 6

Implementace

V teto kapitole je důkladněji popsané zajímavé pasáže a kličkové prvky implementaci monitorovacího systému. Také popsané podobnosti a odlišnosti implementaci serverové a klientové části aplikaci.

Implementační jazyk pro realizaci programu byl vybrán C++ a jeho standart z roku 2017, kvůli některým jeho osobitostem, jako například použiti specifikovaného deleteru u smart pointerů.

6.1 Společná část

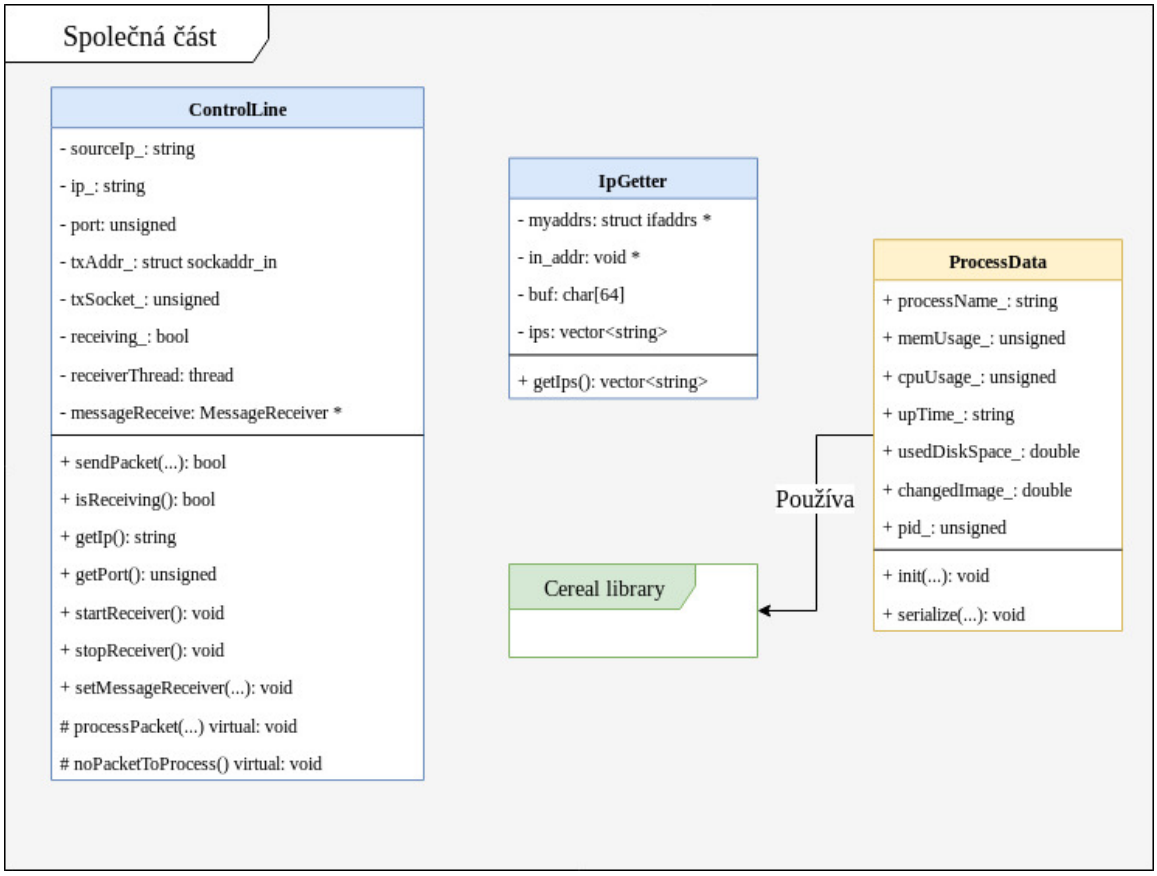
Server a klient mají některé společné implementační detaily, jako komunikační klient, základ kterého mají obě části stejné, protože každá z nich inicializuje sokety, připojuje k multicastové skupině atd. Takže jedná společná část je **ControlLine**, nebo třída pro zajištění síťové komunikaci. Rozdíl mezi klientem a serverem je v tom, že server pouze poslouchá na multicastové adrese, a klient na stejnou adresu posila data a nic nepřijma. Proto server má aktivovaný přijímač (Receiver) zprav, a klient vysílač (Transmitter).

Ještě jedná shoda mezi serverem a klientem je v tom, že každý z nich musí vědět svoje IP, aby měli možnost se připojit na multicast. Proto oni oba používají třídu **IpGetter**, která umí načítat a vracet všechny IPv4 adresy, které má počítač, na kterém běží patřičný program.

Navíc obě části jak klientova tak i serverová používají pro uchovávaní informaci o procesu strukturu **ProcessData**. Tato struktura má v sobě veškerou sledovanou informaci jako ID procesu, jeho název, požitou operační paměť a procesorový čas, čas běhu procesu atd. Ale server navíc má nadstavbu nad teto strukturou, která uchovává některou informaci navíc. Také textbfProcessData používá knihovnu **Cereal**, aby se dalo serializovat a deserializovat data v ní, aby poslat přes síť a rekonstruovat data z ní na jiném místě.

Pro komunikaci klient a server používají zprávy, na začátky bylo vyřešeno že typů zprav bude tři: informační, registrační a notifikační. Ale byl implementován jenom informační typ zpráv *INFO* pro monitorovaní procesu a *STATION_INFO* pro monitorovaní parametrů cele stanice. Ale server už je připraven na dva další typy, *REGISTER* a *KEEP_ALIVE*, které by měli říkat o registraci klienta na serveru a o jeho živosti. Oni nebyli využity původně s důvodu malého intervalu posílaní zprav, a navíc z toho důvodu, že z času příchodu zprávy na server lze jednoduše říct jak o živosti tak i o registraci klienta.

Diagram tříd pro společnou čast projektu je na obrazku 6.1.



Obrázek 6.1: Společná část klienta a serveru. (Modrou barvou jsou označené třídy, žlutou – struktury, zelenou – knihovny.)

6.2 Server

Server je zvláštní aplikaci, spouští se samostatně. Připojuje se na zadanou multicastovou skupinu přes zadaný port a přijímá data od všech klientů ve stejné skupině. Kromě toho, že server umí přijímat zprávy od klienta a zobrazovat je, on zobrazuje varování o překročení kritických hodnot, nastavovaných v souboru *configuration.cfg*. Tyto varovaní vypisují typ chyby, informace o chybě, z jakou stanici nebo procesem tato chyba spojena a čas, kdy varovaní se zobrazilo. Server má následující nápovědu pro start:

```
andrei@DMN:~/Dropbox/bachelor_work$ ./server
USAGE:  server MULICAST_ADDRESS MULTICAST_PORT

        MULICAST_ADDRESS - address of multicast group where clients send packets
        MULTICAST_PORT - port for communication
andrei@DMN:~/Dropbox/bachelor_work$
```

Obrázek 6.2: Nápověda pro spuštění serveru

Server má určité odlišnosti od klienta. Tak on plouživá svojí strukturu pro uchovávaní informaci o procesech ze stanici **ServerProcessData**.

ServerProcessData je strukturou poděděnou od **ProcessData**. Takovým způsobem lze jednoduše rozšířit evidovanou informaci o své vlastnosti. Mapu těchto struktur drží v sobě třída **StationData**.

StationData – je třída pro uchovávaní informaci o cele stanici, včetně všech monitorovaných procesu teto stanici. S toho důvodu ona, kromě informaci o procesech, má údaje o IP stanici, času poslední zprávy, část použitého místa na disku a procent toho, jak se změnila obrazovka. Mapa IP všech stanic a těchto tříd je součásti třídy **MessageReceiver**.

MessageReceiver – třída, která umí zpracovávat zprávy z multicastu od klientů, parsovát informaci z nich a uchovává tyto udaje v jednom místě. Pomoci teto třídy lze přistupovat k informaci o stanicích podle IP jednotlivých stanic nebo získat udaje o všech pracovištích.

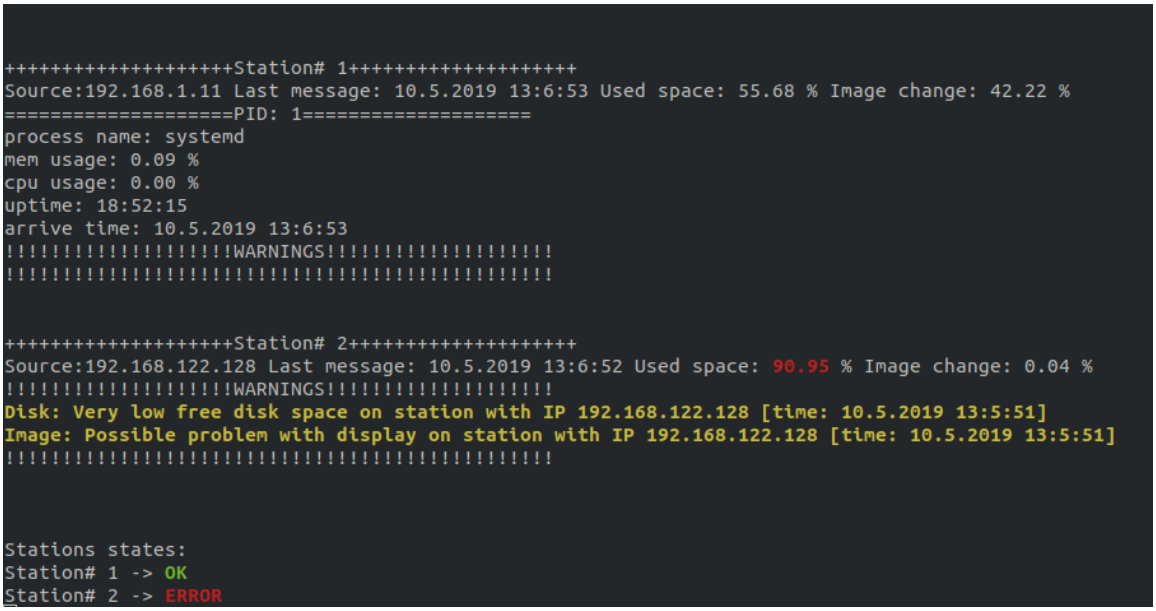
StationParser – třída pro načtení a parsování konfiguračního souboru *configuration.cfg*, ve kterým jsou kritické hodnoty pro monitorované klientem položky.

Tento soubor má následující strukturu s popiskem všech použitých položek:

```
#minimal acceptable process CPU usage in percentage of whole system CPU usage
CPU_minimum = 0
#maximal acceptable process CPU usage in percentage of whole system CPU usage
CPU_maximum = 80
#minimal acceptable process virtual MEMORY usage in percents of whole virtual memory
MEM_minimum = 0
#maximal acceptable process virtual MEMORY usage in percents of whole virtual memory
MEM_maximum = 20
#maximal acceptable DISK memory occupancy in root folder in percents
disk_space_maximum = 80
#minimal acceptable image change percentage
image_change_minimum = 0
#maximal acceptable image change percentage
image_change_maximum = 60
#maximal acceptable delay of message from client in MINUTES
#if server would not receive message in this interval, warning will be shown
message_delay_maximum = 2
#period of time in minutes, when old warning will be erased
warning_delete_period = 60
```

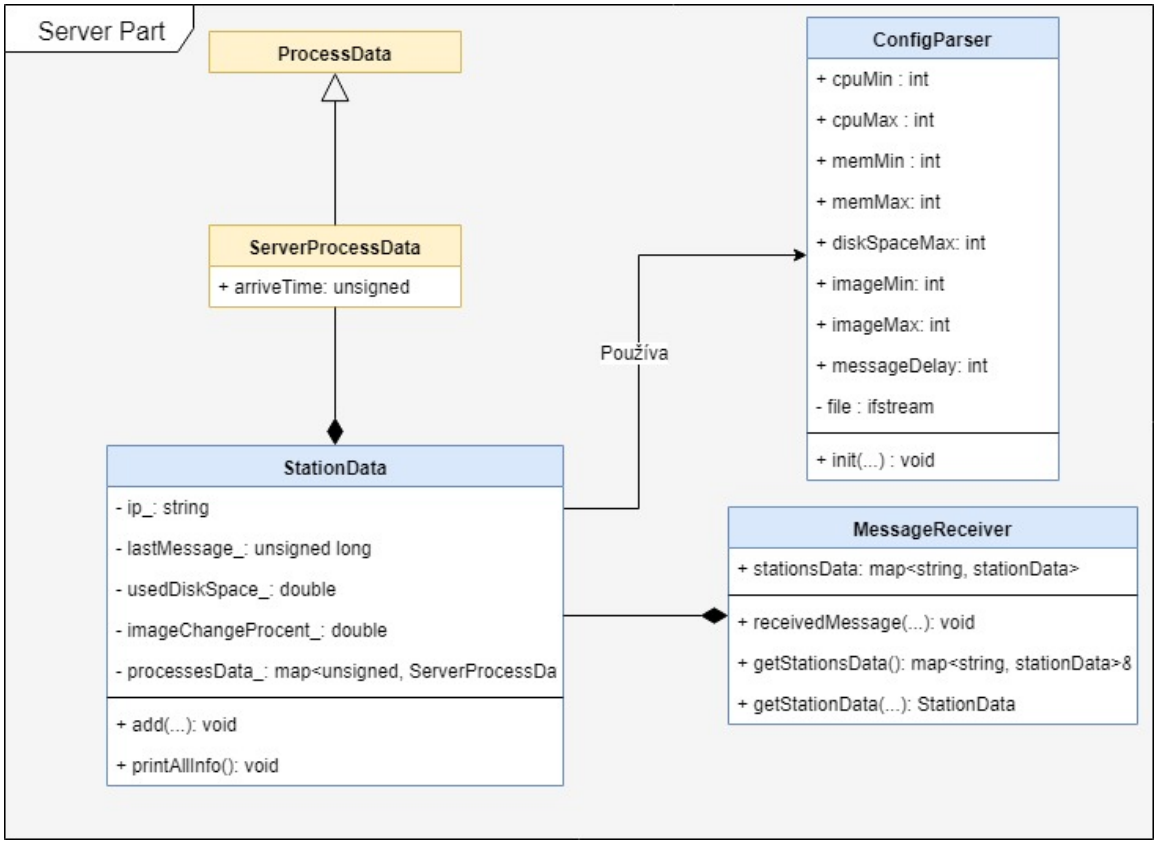
Obrázek 6.3: Příklad konfiguračního souboru pro server.

Výstup serveru se skládá z IP sledované stanici, času poslední zprávy od klienta (veškerý čas je v UTC), který sleduje stav celé stanici, kolik místa je obsazeno ve kořenovém adresáři, a procent o kolik se změnila obrazovka. Pak nasleduje informace o jednotlivých sledovaných procesech. PID a název procesu, jakou část virtuální operační paměti využívá proces, jakou část CPU od procesorového času celého systému využívá proces, UPTIME procesu a čas příchodu zprávy od klienta. Pak nasleduje blok s varováními, které se vyskytli na patřičné stanici. Po informaci o všech stanicích nasleduje jednoduchý výpis stavů stanic, který popisuje, jestli jsou na nějaké stanici varovaní. Příklad výstupu je na obrázku 6.4. Tyto výpisy se automatický obnovují při příchodu nových zprav a kromě toho každých 10 sekund.



Obrázek 6.4: Příklad celého výstupu serveru

Diagram tříd pro serverovou část aplikaci je na obrázku 6.5.



Obrázek 6.5: Serverová část aplikaci. (Modrou barvou jsou označené třídy, žlutou – struktury.)

6.3 Klient

Klient je nejdůležitější prvek celého monitorovacího systému, protože on používá veškeré externí knihovny, komunikuje s operačním systém pro sběr dat a se serverem pro jejích analýzu a zobrazení.

Klient jako server je zvláštní aplikaci. Spouští se z parametry pro komunikaci přes multicast a to jsou adresa multicastové skupiny a komunikační port. Třetím nepovinným parametrem je ID procesu, který bude sledován tímto klientem, jestli parametr nezadán, klient bude monitorovat obsazené místo na disku a sledovat obrazovky stanice. Klient má následující parametry spouštění:

```
andrei@DMN:~/Dropbox/bachelor_work$ ./client
USAGE:  client MULICAST_ADDRESS MULTICAST_PORT [PID]

        MULICAST_ADDRESS - address of multicast group where server listen to clients
        MULTICAST_PORT - port for communication
        PID - ID of process to be monitored, if this argument is not given, state of the station will be monitored
andrei@DMN:~/Dropbox/bachelor_work$
```

Obrázek 6.6: Nápověda pro spuštění klienta

Hlavní třídou, kterou používá klient je **ProcessHandler**. Ona se zabývá sbíráním informaci o systému a procesu, vytvořením screenshotů, jejích porovnáním, serealizaci dat do

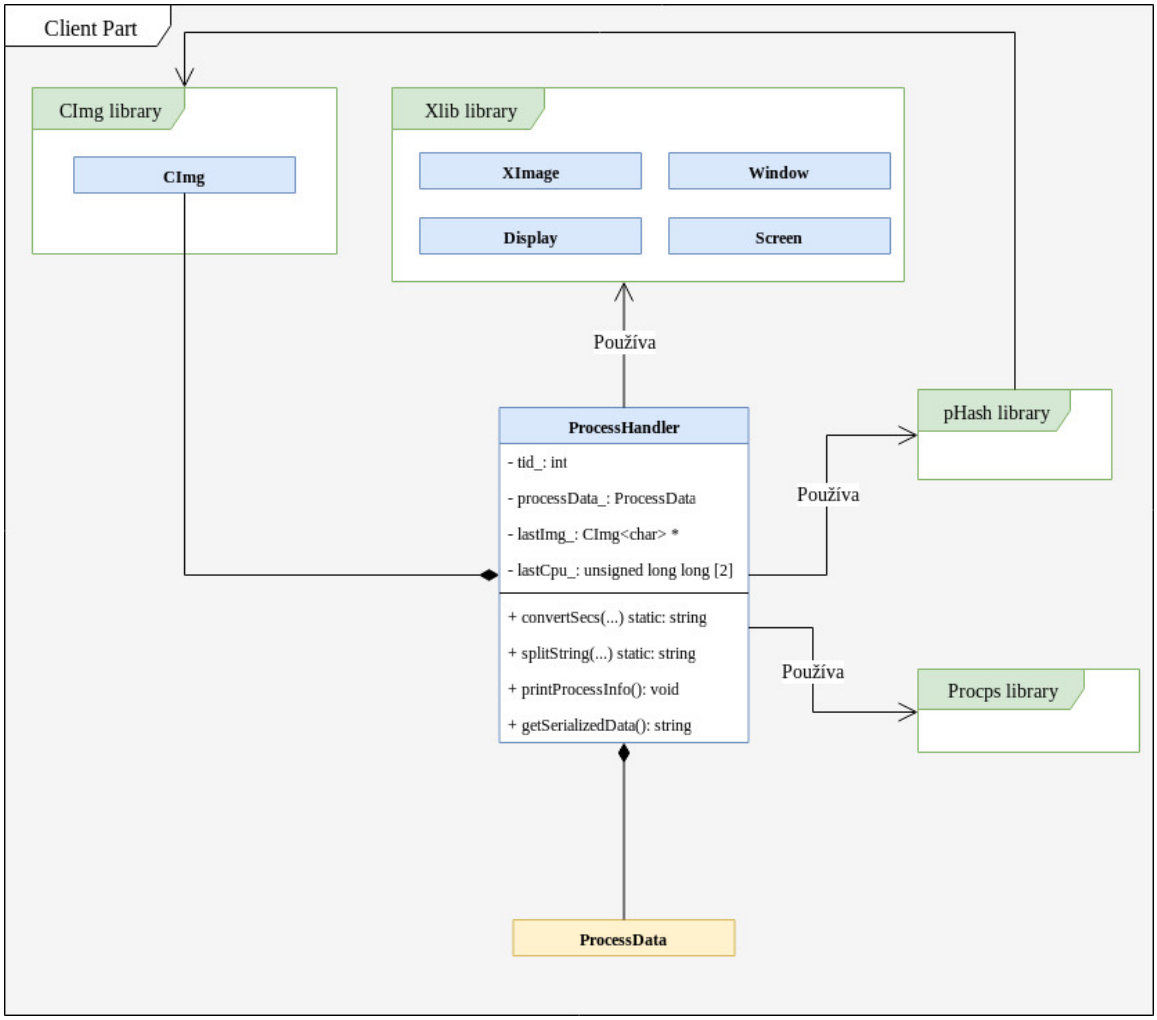
formátu řetězce pro následující odesílaní na multicast. Informace o procesu tato třída drží v struktuře **ProcessData**.

Jedná z knihoven kterou používá klient je systémová knihovna **Xlib**, pomoci jí lze snímat celou obrazovku nebo jenom určité okno. Pak screenshot obrazovky má byt převáděn do formátu obrázku z knihovny **CImg**, protože **XImage** a **CImg** mají různá uložené RGB hodnoty pixelů obrazovky. Pak po transformaci se používá knihovna **pHash**, aby vypočítat percepční hashy aktuálního a předchozího screenshotů a porovnat jé. Pomoci porovnávací funkci lze zjistit jak se liší dvě obrazovky a podle toho dozvědět procent o jaky se obrazovka změnila. A tento procent odeslat serveru. Použití teto knihovny je efektivnější než porovnávaní každého pixelů obrazovek a přesnější než obyčejné kryptografické hashy.

Také důležitou knihovnou je knihovna **procps**, která poskytuje rozhraní pro načítaní informaci o systému a procesech, s použitím Linuxového adresáře */proc*. Pomoci teto knihovny načítám informaci o čase běhu operačního systému ze struktury **sysinfo**. Pak pomoci teto informaci a informaci o startu aplikaci ze struktury *proc_t* lze zjistit UPTIME procesu. Navíc pomoci teto stejné struktury lze zjistit použitou operační paměť konkretního procesu a název exekučního souboru toho procesu.

Klient čte data o stanice každé tři vteřiny a pak posila načtenou informaci na server. V některých situacích může dojít k určitému časovému posunuti příchodu dat na server, a to může byt kvůli nízkému výkonu počítače nebo zpoždění na síti nebo jiné. Tyto případy neznamenají, že nastala nějaká vnitřní chyba klientu, nebo aplikaci, a proto ve výpisu přijatých dat na serveru existuje pole *arrive time*. Podle kterého lze zjistit čas příchodu informaci o procesu nebo stanici.

Diagram tříd pro klientovou část aplikaci je na obrázku 6.7.



Obrázek 6.7: Clientová část aplikaci. (Modrou barvou jsou označené třídy, žlutou – struktury, zelenou – knihovny.)

6.4 Ověření funkčnosti aplikaci

Způsobem testovaní funkčnosti části aplikaci byli *unit testy*. Pomoci takových testů lze jednoduše zkontrolovat správnost jednotlivých modulů zdrojového kódu programu. Z důvodu toho, že klient a server mají dostatečný počet společných části, veškeré *unit testy* jsou uležené v jednom hlavičkovém souboru **tests.h**. Testy jsou napsané pro některé metody tříd nebo ověřují nějakou malou funkcionalitu projektu. Testy sdělují jeden jmenný prostor a podle nazvu funkci testu lze jednoduše zjistit jakou částku testují. V projektu jsou následující *unit testy* :

- **sendStringsTest** – ověřuje správnost odesílaní řetězců na multicastovou adresu pomoci třídy *ControlLine*.
- **testProcess** – testuje jednu z hlavni funkcionalit monitorovacího klienta, a to sbíraní a vypisovaní informaci třídou *ProcessHandler*.

- **sendProcessInfo** – testuje se několik věci dohromady, a to cely cyklus běhu klienta. Sběr dat o procesu a operačním systému, jejích serializace a odesílaní na multicast.
- **takeScreenshot** – testuje se funkcionalita snímaní screenshotů obrazovky a převod do formátu obrázku knihovny *CImg*.
- **imageLibTest** – s využitím testu pro snímaní screenshotů, testuje se knihovna *pHash* a její funkcionalita porovnaní dvou obrázků. Screenshoty se snímají s určitou časovou pauzou.
- **imageLibTest2** – test velice podobný na předchozí, ale první screenshot se snímá automaticky a pak program čeká na vstup od uživatele. Tímto způsobem uživatel může připravit obrazovky nejvhodnějším pro testovaní způsobem.
- **twoImageTest** – ještě jeden test knihovny *pHash*, který očekává na vstup cestu k dvěma obrázkům a porovnává jejích hashy.
- **runTests** – test, který spouští několik vybraných testů. Může sloužit pro automatické spouštění všech *unit testů*.

Kapitola 7

Testování

Na začátku kapitoly je popsán způsob jak se da vyzkoušet monitorovací systém. Také dal je popsáno jak probíhalo testovaní správností sledovacího systému, jak on byl nasazen a vyzkoušen v praxi, na jakých systémech běžela aplikace, uvedený některé výsledky testovaní a na konci kapitoly ohodnocen výsledný stav aplikaci vzhledem k provedenému testovaní.

Aby zkompilovat program stačí jenom spustit příkaz *make* nebo *make all* ve složce s souborem Makefile. Toto mělo by nakonfigurovat veškeré knihovny třetích stan, zkompilovat je a pak udělat build klientu a serveru. Po úspěšné kompilaci aplikaci měli by vzniknout dva spustitelné soubory *client* a *server*. Ve výjimečných případech v systému můžou chybět knihovny potřebné pro kompilaci a běh programu. V tom případě chybějící knihovny měli by byt nainstalovány uživatelem manuálně.

Pro vlastní testovaní aplikaci lze využit jeden počítač s virtuální mašinou nebo několik počítačů ve stejné síti. Na jednom z systému má byt spouštěn server 6.2 s vhodnou pro komunikaci multicastovou adresou a portem. Ve stejné složce se serverem má nacházet konfigurační soubor *configuration.cfg* s nastavením kritických hodnot monitorovaných položek. Na ostatních systémech měli by byt spouštěný klienti, i z tím, že několik klientů můžou běžet na stejném počítači a monitorovat různé procesy a jeden z nich sledovat stav stanici. Každý klient měl by používat stejnou multicastouvou adresu a port jako server. Pak server měl by začít zobrazovat data od každého z klientů.

Testování komunikaci klientu a serveru probíhalo hlavním způsobem na jednom počítači a ve dvou operačních systémech virtuálním a reálném. Hlavní distribuci pro ověření funkčnosti byla **Ubuntu**, ale testovalo se i na dalších operačních systémech jako **Gentoo**, **Fedora** a **CentOS**. Pro soukromé použiti a rychle spouštění klienta byl napsaný jednoduchý *bash* skript:

```
#!/bin/bash


if [[ ! -z "$1" ]]
then
        var=$1;
else
        echo "Set the PID of program"
        exit -1;
fi
```

```
#use to test on real machines
./client 239.0.0.1 26001 $var

#use to test on virual machine
#./client 224.0.0.1 26001 $var
```

Při osobním testování klienta, pro ověření že on se skutečně připojuje na multicast, odesílá tam informaci a jaké data posílá, byl použitý nastroj pro monitorovaní síťového provozu **Wireshark**. Pro testování funkčnosti klienta bez možnosti spouštění serveru, v kódu jsou různá debagovací výpisy, makra a metody. Tak nejduležitějším makrem pro testovaní je jednoduché logovací makro, které vypisuje název souboru, řádek volaní makra, název funkce a zprávu, vypadá takto:

```
#ifdef DEBUG
#define LOG(msg) \
    std::cout<<__FILE__<<'['<<__LINE__<<"] "<<__func__<<"(): "<<msg<<std::endl
#else
#define LOG(msg)
#endif
```

Integrační testování proběhlo tak, že na virtuální mašině byl spuštěn klient, který monitoroval běh nějakého náhodného procesu v operačním systému. A na reálném počítači běžel server, který čekal na zprávy od klienta a vypisoval veškerou získanou informaci. Příklad serverového výstupu při takovém testovaní je na obrázku 7.1. Zobrazena část výstupu, která obsahuje podrobnější informaci o stanicích a procesech.

```
++++++++++++++++++++Station# 1++++++++++++++++++++
Source:192.168.1.11 Last message: unknown Used space: unknown % Image change: unknown %
====================PID: 14823====================
process name: qemu-system-x86
mem usage: 26.99 %
cpu usage: 1.17 %
uptime: 0:17:59
arrive time: 27.4.2019 17:25:48
!!!!!!!!!!!!!!!!!!!!WARNINGS!!!!!!!!!!!!!!!!!!!!
CPU: Critical CPU usage on station with IP 192.168.1.11 with process PID: 14823 [received at 27.4.2019 17:25:39]
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


++++++++++++++++++++Station# 2++++++++++++++++++++
Source:192.168.122.128 Last message: 27.4.2019 17:25:47 Used space: 90.95 % Image change: 26.27 %
====================PID: 1474====================
process name: gnome-shell
mem usage: 12.35 %
cpu usage: 0.00 %
uptime: 0:13:30
arrive time: 27.4.2019 17:25:48
====================PID: 2574====================
process name: python3
mem usage: 23.67 %
cpu usage: 0.00 %
uptime: 0:09:32
arrive time: 27.4.2019 17:25:49
!!!!!!!!!!!!!!!!!!!!WARNINGS!!!!!!!!!!!!!!!!!!!!
CPU: Critical CPU usage on station with IP 192.168.122.128 with process PID: 1474 [received at 27.4.2019 17:25:39]
CPU: Critical CPU usage on station with IP 192.168.122.128 with process PID: 2574 [received at 27.4.2019 17:25:40]
Disk: Very low free disk space on station with IP 192.168.122.128 [received at 27.4.2019 17:25:38]
Image: Possible problem with display on station with IP 192.168.122.128 [received at 27.4.2019 17:25:44]
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Obrázek 7.1: Část výstupu monitorovacího serveru při testování. Dva klienty běželi na stejném počítači a monitorovali dva různé procesy. Třetí klient běžel na druhém počítači a monitoroval svůj proces.

Pro simulaci zátěže operační paměti, procesoru, vstup-výstupních operaci atd. v Linux existují různé nastroje. Například program **stress**[1], který napsán v jazyce C a je jednoduchým generátorem pracovního zatížení pro systémy POSIX. Baliček teto aplikaci existuje pro většinu UNIX operačních systému a je velice jednoduchý na použiti. Při tom umí simulovat chyby skoro všech vybraných pro sledovaní komponent systému. Také pro simulaci zátěži procesoru a paměti byli implementovány jednoduché skripty. Teto scripty se nacházejí ve složce *test_dir* projektu a při jejich spuštění bez parametru do konzole se vypíše nápověda pro jejích použití. Pro simulaci aktivity na obrazovce stačí jenom hýbat oknem s nějakým programem, hlavním cílem je to aby se dost měnila obrazovka na monitoru a tyto změny pak všimne klient a odešle na server.

Monitorovací aplikace byla nasazena v praxi při testovaní systému řízení letového provozu, a konkretně radaru a pak i VCS (voice communication system). Za několik hodin monitorovaní neboli odhalené žádné zásadní problémy v monitorovací aplikaci ani v sledovaných softwarech. Tak že z toho lze dospět k závěru, že implementovaný program může byt využity pro reálné testovaní softwaru.

Jedním z hlavních problémů, které byli odhalené při testovaní, bylo poměrně velké zatížení procesoru monitorovacím klientem, který sleduje stav obrazovky a využité místo na disku stanici. Důvodem k takovému zatížení je především časté vytvoření screenshotu obrazovky, vypočítaní jejích hashů a jejích porovnaní. Tento problem nejde lehce vyřešit, protože vytvoření screenchotů a jejích porovnaní dělají knihovny *Xlib* a *pHash*. Jedinou věc kterou se mi podařilo trochu optimalizovat a tím snížit zátěž procesoru, je převod obrázku z formátu *Xlib* do formátu *CImg* pomoci makra druhé knihovny.

Také problém může vzniknout při testovaní klientů na počítači bez monitoru, což se v praxi nemělo by stát. Ale v případě když monitorovací aplikace je spuštěna přes nějaký SSH klient, tak on měl by byt spuštěn s tak zvaným „X11 forwarding“ aby se použil monitor/displej toho, kdo se připojuje k počítači.

[1] https://people.seas.harvard.edu/~apw/stress/

Kapitola 8

Závěr

Cílem této práce bylo nastudovat současný stav ve sféře monitorovaní aplikaci. Následně navrhnout a implementovat novou monitorovací aplikaci se záměrem na sféru řízení letového provozu. Aby se jí dalo lehce použit s aplikacemi v teto sféře i při tom aby ona se něčím lišila od už existujících rašeni. Pak vytvořené řešení otestovat a ohodnotit.

Při seznamování s monitorovacími systémy obecného formátu i formátu speciálního, jako aplikaci existující přímo pro sledovaní běhu systémů řízení letového provozu, byly odhalené jejích výhody a nedostatky. Na základě teto analýzy a požadavků pro monitorovací systém se vytvořil návrh aplikaci typu klient-server, kde sledováním stavu operačního systému a vybraných procesů se bude zabývat klient, spuštěny na konkretním počítači, pro monitorovaní konkretního procesu. A sběrem, zobrazením a připadnou analýzou – server.

Výsledná aplikace byla implementovaná v jazyce *C++* s použitím některých knihoven třetích stran. Kromě monitorovaní důležitých vlastnosti operačního systému a samotného procesu (jako využívaná procesem operační paměť, využívaný procesorový čas, čas běhu procesu, použité prostor na disku), je monitorován i stav obrazovky. A to tak, že jednou za čas se snímá screenshot cele obrazovky, vypočítavá se její percepční hash a porovnává se s hashem předchozího screenshotu. Takovým způsobem lze poměrné přesné odhalit rozdíly mezi obrazovkami a při málem procentu lze říct, že se na obrazovce nic nemění, a že software s velkou pravděpodobnosti funguje špatně.

Kromě implementaci samotného monitorovaní, byli implementovaný některé skripty, pomoci kterých lze jednoduše simulovat zátěž systému.

Testovaní se ukázalo, že funkčnost aplikaci je na dostatečné vysoké úrovní i při tom, že se objevily některé nedostatky aplikaci ve spotřebě systémových zdrojů.

Co se tyká možného dálnějšího vývoje projektu, existuje několik věci, které by mohli byt vylepšený nebo případně změněny. Jedná z těch věci je různé typy zprav od klienta k serveru. Server už je připraven na některé další typy zpráv a můžou byt přidávaný i vlastní. Také v budoucnosti mohlo by byt optimalizováno vytvoření screenshotu obrazovky a jejích porovnaní, aby používalo miň CPU a bylo rychlejší. Také mohli by byt přidány další monitorovací vlastnosti podle potřeby, kód projektu je dostatečné flexibilní, a proto modifikace takového charakteru neměli působit problémy. Navíc možným vylepšením bude rozšíření funkcionality serveru, a to může byt jiný způsob zobrazení informaci od klientů (např. přes webové rozhraní), hlubší analýza sbíraných dat, ukládaní těch dat do databáze, posílaní notifikaci pro ředitele stanic při výskytu chyb atd.

Literatura

[1] Cantrill, B.; Leventhal, A.: About DTrace. [Online; navštíveno 12.03.2019].
URL http://dtrace.org/blogs/about/

[2] cereal - A C++11 library for serialization. [Online; navštíveno 21.03.2019].
URL https://uscilab.github.io/cereal/

[3] What is the CImg Library ? [Online; navštíveno 20.03.2019].
URL http://cimg.eu/reference/group__cimg__faq.html#ssf11

[4] Learn C++ programming language. [Online; navštíveno 19.03.2019].
URL https://www.tutorialspoint.com/cplusplus/cpp_tutorial.pdf

[5] Eigler, F. C.: Systemtap tutorial. [Online; navštíveno 11.03.2019].
URL https://sourceware.org/systemtap/tutorial/1_Introduction.html

[6] Gregg, B.; Maur, J.: *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*. Prentice Hall, první vydání, 2011, ISBN 978-0132091510.

[7] ICZ LETVIS - TCM Technical Control & Monitoring. [Online; navštíveno 14.03.2019].
URL https://www.iczgroup.com/wp-content/uploads/2018/04/ICZ_PL_TRANS_ICZ-LETVIS-TCM_EN_1802_01.pdf

[8] Ltd, T.: M/MONIT User Manual. [Online; navštíveno 11.03.2019].
URL https://mmonit.com/documentation/mmonit_manual.pdf

[9] McCanne, S.: libpcap: An Architecture and Optimization Methodology for Packet Capture. [Online; navštíveno 16.03.2019].
URL https://sharkfestus.wireshark.org/sharkfest.11/presentations/McCanne-Sharkfest'11_Keynote_Address.pdf

[10] Perceptual Hashing. [Online; navštíveno 24.03.2019].
URL http://bertolami.com/index.php?engine=blog&content=posts&detail=perceptual-hashing

[11] pHash The open source perceptual hash library. [Online; navštíveno 20.03.2019].
URL http://www.phash.org/

[12] Why are there so many procps projects? [Online; navštíveno 20.03.2019].
URL https://gitlab.com/procps-ng/procps/blob/master/Documentation/FAQ

[13] Project, T. L.: The LTTng Documentation. [Online; navštíveno 12.03.2019].
URL https://lttng.org/docs/v2.10/

[14] *PS(1) User Commands*. [Online; navštíveno 16.03.2019].
URL http://man7.org/linux/man-pages/man1/ps.1.html

[15] SafeGuard. [Online; navštíveno 14.03.2019].
URL https://www.eizoglobal.com/products/atc/safeguard/index.html

[16] Silberschatz, A.; Galvin, P. B.; Gagne, G.: *Operating System Concepts*. Wiley, 9 vydání, 2012, ISBN 978-1118063330.

[17] Stroustrup, B.: *The C++ Programming Language*. Addison-Wesley Professional, třetí vydání, 2000, ISBN 978-0201700732.

[18] Stroustrup, B.: Evolving a language in and for the real world: C++ 1991-2006. *HOPL III*, 2007, [Online; navštíveno 17.03.2019].
URL http://stroustrup.com/hopl-almost-final.pdf

[19] Wikipedia contributors: Linux Trace Toolkit. [Online; navštíveno 12.03.2019].
URL https://en.wikipedia.org/w/index.php?title=Linux_Trace_Toolkit&oldid=780663282

[20] Wikipedia contributors: LTTng. [Online; navštíveno 12.03.2019].
URL https://en.wikipedia.org/w/index.php?title=LTTng&oldid=875135795

[21] Závodský, K.: *HISTORIE. řízení letového provozu*. Řízení letového provozu ČR/ARTillery, 2014, ISBN 978-80-905939-0-9.

[22] Řízení letového provozu ČR: Letové provozní služby. [Online; navštíveno 10.03.2019].
URL http://www.rlp.cz/sluzby/nase/Stranky/default.aspx

Příloha A

Obsah přiloženého paměťového média

- **app/** – zdrojové kódy aplikaci
  - **include/** – hlavičkové soubory
  - **lib/** – externí knihovny
  - **src/** – zdrojové soubory
  - **test_dir/** – simulační skripty
  - **client** – spustitelný soubor pro klienta
  - **server** – spustitelný soubor pro server
  - **README.md** – návod na použití
  - **configuration.cfg** – konfigurační soubor
- **text/** – zdrojové soubory textu práci
- **xpapla00-monitorovani-aplikaci.pdf** – text praci