STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ

HÁLOVA 16, 851 01 BRATISLAVA

Linux File Organizer

Matej Petuchovský

Bratislava

2026

Ročník štúdia: IV A

STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ

HÁLOVA 16, 851 01 BRATISLAVA

Linux File Organizer

Matej Petuchovský

Bratislava

2026

Ročník štúdia: IV A

Ing. Dominik Zatkalík, PhD

Zadanie koncoročnej odbornej práce

1. Popíšte základy správy súborov v Unix/Linux
2. Analyzujte a porovnajte existujúce nástroje
3. Navrhnite softvérovú architektúru nástroja
4. Implementujte jadro vyhľadávania
5. Implementujte sortovanie podľa pravidiel
6. Implementujte inteligentne čistenie
7. Optimalizujte zdrojový kód aplikácie
8. Zabezpečte kvalitu a spoľahlivosť pomocou unit testou
9. Pripravte distribúciu a dokumentáciu
10. Navrhnite model propagácie a udržateľnosti softvéru

Čestné vyhlásenie

Vyhlasujem, že prácu stredoškolskej odbornej činnosti na tému „Linux File Organizer“, som vypracoval samostatne, s použitím uvedených literárnych zdrojov. Prácu som neprihlásil a ani neprezentoval v žiadnej inej súťaži, ktorá je pod gestorstvom MŠVVaM SR. Som si vedomý dôsledkov, ak uvedené údaje nie sú pravdivé.

.....

V Bratislave, <dd. mm. rrrr> Matej Petuchovský

Poďakovanie

Rád by som sa touto cestou poďakoval svojmu Ing. Dominiku Zatkalíku, PhD., za prístup a odborné rady. Tiež by som sa rád poďakoval spoločnosti 2Ring s.r.o. za odbornú pomoc a podporu pri realizácii praktickej časti mojej práce, najmä za vedomosti a skúsenosti, ktoré som mal možnosť získať počas spolupráce.

Abstrakt

Tento projekt sa zameriava na návrh a vývoj konzolového nástroja pre organizáciu súborov v adresári v operačnom systéme Linux. Riešenie poskytuje funkcionalitu vyhľadávania, sortovania a čistenia z možnosťou rosných nastavení. Veľká pozornosť bola udelená rýchlosti a spoľahlivosti nástroja. Práca tak tiež obsahuje analýzu existujúcich podobných nástrojov a možný model propagácie a údržby.

Kľúčové slova: CLI, Linux, Rust, Súborový systém

Abstract

This project focuses on the design and development of a CLI tool for working with files and directories in the Linux operating system. The solution provides the functionality of searching, sorting and cleaning from the possibility of dew settings. Great attention was paid to the speed and reliability of the tool. Thus, the work also includes an analysis of existing similar tools and a possible model of promotion and maintenance.

Keywords: CLI, Linux, Rust, file system

Obsah

Zoznam skratiek, značiek a symbolov

GUI	Grafické používateľské rozhranie
CLI	Rozhranie príkazového riadku
BFS	Vyhľadávanie v šírku
DFS	Vyhľadávanie v hĺbku
Regex	Regulárne výrazy
VCS	Systém správy verzii
Logovanie	Schopnosť procesu zaznamenať nejakú informáciu pre neskoršiu analýzu
Env	Environmentálne premenne
Commit	Uložený záznam v VCS git
Heap	Halda
Stack	Zásobník
Crate	Balík v ekosystéme Rust

ÚVOD

Operačný systém Linux namiesto riadenia počítača pomocou vylúčene grafickým používateľským rozhraním (ďalej GUI), ponúka aj rozhranie príkazového riadku (ďalej CLI). Hlavným riadiacim CLI v Linuxe je konzola. Funkcionálne možnosti konzoly sa zväčša definujú množstvom a možnosťami jej príkazových nástrojov, pritom každý používateľ vie sam vybrať nástroje, ktoré bude mať dostupné v konzole.

Takých nástrojov je veľa, ale oni nie vždy spĺňajú všetke očakávanie používateľa. Používateľ v svojom rade je motivovaný komponentovou architektúrou systému Linux vyberať zo spĺňajúcich podobne funkcie nástrojov tie, čo najviac vyhovujú jeho osobným požiadavkám.

Jednou z oblasti v ktorej existuje veľa odlišných požiadaviek, a následkom toho aj nástrojov, je manažovanie súborového systému. Častou požiadavkou na také nástroje je jednoduchosť použitia, keďže najrozšírenejšie CLI nástroje časom nadobúdajú veľké množstvo funkcií a stavajú sa komplikovanými v použití. Tak tiež často je žiadaná vysoká rýchlosť nastroja, aby nezväčšovať aj tak časovo náročne operácie nad súbormi. A samozrejme je žiadane aby taky nastroj bol spoľahlivý, aby kvôli chybovosti v kóde nastroja nedošlo k poškodeniu súborov a dat v nich.

Cieľom tejto prace je navrhnúť, implementovať a overiť funkcionalitu konzolového nastroja v operačnom systéme Linux, určeného na organizáciu súborov v adresári, spĺňajúceho často žiadane vlastnosti, medzi, ktoré patrí jednoduchosť použitia, rýchlosť a bezpečnosť.

1 PROBLEMATIKA A PREHĽAD LITERATÚRY

V tej to kapitole sumarizujeme teoretické poznatky a existujúce prístupy k sprave súborov a adresárov, ktoré sú potrebné pri návrhu, tvorbe a údržbe nastroja. Pozornosť je venovaná špecifike súborového systému v operačnom systéme Linux, pojmom CLI a CLI nastroj, informácie o spôsoboch riešenia základných úloh nastroja a technologické informácie súvisiace s údržbou projektov v štádiu vývoja. Cieľom prehľadu je vytvoriť súvislý teoreticky základ, ktorý slúži, ako východisko pre ďalšie časti práce.

1.1 PRÍKAZOVÍ RIADOK A CLI NÁSTROJE

Rozhranie príkazového riadku, alebo CLI, je spôsob ovládania počítača pomocou textových príkazov zadávaných z klávesnice. V systéme Linux sa pod CLI primárne mysli najmä práca z shell. To je program, ktorý prijíma klávesnicové príkazy a odovzdáva ich operačnému systému na vykonanie. Pri tom v Linuxe často používajú CLI a GUI naraz. V takom prípade na prístup k CLI je potrebný program, ktorý sa nazýva konzolový emulátor alebo zjednodušene konzola. CLI sa považuje za veľmi výrazný spôsob komunikácie s počítačom. [1]

Každý príkaz v CLI predstavuje sebou samostatný program, ktorý voláme CLI nastroj. Funkcionálne možnosti príkazového riadku sa určujú množstvom nainštalovanie konzolových nástrojov. Bežný Linux typický obsahuje doslova tisíce programov použiteľných s príkazového riadku.

1.2 ZÁKLADNE POJMI SÚBOROVÉHO SYSTÉMU

Práca CLI nástrojov, ktoré vyhľadávajú alebo spracúvajú súbory a adresári, opierajú sa na rovnaký základ pojmov: čo je adresát a súbor, aké vlastnosti oni máju, kde sa nachádzajú a, ako vieme k nim pristupovať. Tak tiež často sa vyskytujú problematika rozdielov operačných systémov Linux a Windows. Preto v nasledujúcich kapitolách zavádzame kľúčové definície a pojmi týkajúce sa súborového systému a stručne popíšeme rozdiely, ktoré môžu ovplyvniť prácu z nim v rôznych systémoch.

1.3 SÚBOROVÝ SYSTÉM A JEHO KOMPONENTY

Súborový systém predstavuje sebou spôsob organizácie a prístupu k dátam na zariadení. On ponuka jediný interface pre prácu uloženými objektami. Pre CLI nástroje to znamená tri základne možnosti – možnosť prehľadu uložených objektov, možnosť získania ich metadáta a z pomocou posledných vykonať operácie nad objektami.

V aplikovanom zmysle, súbor je pomenovaný objekt dát, a adresár (alebo priečinok) je špeciálnym súborom-kontajnerom, ktorý obsahuje zoznam mien a reálnych polôh v pamäti iných objektov, buď súborov alebo iných priečinkov. Síce každý priečinok je zároveň súborom, niekedy sa priamo pomenúvajú oba objekty, keďže nie každý súbor je adresárom.

Funkcia väčšiny CLI nástrojov spočíva v prehliadaný stromu adresárov a spracovávaniu objavených súborov na základe ich metá dát. Najčastejšie metá dáta, využívané CLI nástrojmi sú:

- Meno súboru – reťazec v menovom priestore adresára
- Veľkosť – zvyčajne v bytoch
- Časove pečiatky – je ich viacero, príkladom je dátum a čas vytvorenia súboru
- Rozšírenie – konvencia pomenovania rôznych typov súborov

Tu treba spomenúť dôležitú špecifiku názvu súborov v Unix podobných systémoch: „Názvy súborov, ktoré začínajú bodkou, sú skryté. To len znamená, že bežne sa nebude uvádzať, pokiaľ priamo nebude povedané rátať aj zo skrytými súbormi.“ Preto pre CLI nástroje je štandardom zachovať také spravovanie v svojich výstupoch. Robí sa to štandardizovaný parametrom: „-a, --all – zoznam všetkých súborov, dokonca aj tých s názvami, ktoré začínajú bodkou, ktoré zvyčajne nie sú uvedené (skryté).“ [1]

1.3.1 HIERARCHIA ADRESÁROV A CESTY

Adresáre tvoria stromovitú štruktúru: od koreňa, ktorý je najvyšším adresárom, po listy, ktoré predstavujú sebou súbory. V tomto kontexte cesta – je adresou adresára v strome. V Unix podobných systémoch rozlišujú sa absolútne a relatívne cesty.

Kľúčová idea absolútnej cesty spočíva v tom, že ona presne zadáva polohu súbora v strome: „Absolútna cesta začína koreňovým adresárom a nasleduje vetvu stromu vetva, kým sa nedokončí cesta k požadovanému adresáru alebo súboru.“ [1]

Relatívna cesta sa interpretuje vzhľadom na aktuálny pracovný adresár, a nie koreňový adresár. Na navigáciu po strome sa využívajú špeciálne znaky: „"." zápis odkazuje na pracovný adresár a zápis ".." sa vzťahuje na nadradený adresár“ [1]

Praktický záver s toho pre CLI nástroje znamená, že rovnaká logika navigácie v súboroch ma vedieť vysporiadať sa aj z relatívnymi aj s absolútnymi cestami, aby nedošlo k neočakávaným pre používateľa výsledkam.

1.3.2 ROZDIELY MEDZI WINDOWS A LINUX

Kľúčové pre nás rozdiely sa začínajú s rozdielu absolútnych ciest a deliaceho symbola. V Windows systémoch úplná cesta k súboru sa začína z písmena diska (napríklad, „C:“), a deliaci symbol je spätná lomka. V systémoch podobných Linux je model iný – v ceste nie sú žiadne písmená jednotiek a vedúca lomka označuje koreň súborového systému. Bez vnímania týchto rozdielov, program napísaný na multiplatformovom jazyku nebude správne fungovať na oboch systémoch. [1]

1.4 TEÓRIA PREHĽADÁVANIA ADRESÁROV

Pred tým, ako vykonať akcie nad súbormi, treba najprv vyhľadať vyhovujúce súbory v adresári. Keď, že často adresári máju vysoké hniezdenie, to môže byť ne úplne triviálne. V nasledujúci podkapitolách zadefinujeme čo je prehľadávanie, uvedieme dve základne stratégie a rozoberieme, ako vieme vyhodnotiť či nám súbor vyhovuje.

Pod prehľadávaním adresa sa obyčajne mysli získanie zoznamu obsahu adresára, a v prípade potreby aj zoznam obsahu všetky vložených adresárov. Keď získavame zoznámi aj vložených adresárov, voláme to rekurzívnym prehliadaním.

Veľkosť stromu priamo určuje náklady na vyhľadávanie. Na veľkých stromoch hlavná záťaž zvyčajne ide skôr od operácií súborového systému

(prístup k metá dátam, kontroly práv a pod), než do „čistého“ spracovania reťazcov. V praxi, náklady na prehliadanie sú približne úmerné počtu navštívených uzlov stromov a počtu prečítaných záznamov. Samozrejme dôležite je prejsť každý uzol iba raz, na čo sa väčšinou používa jedna s dvoch stratégii: BFS alebo DFS. [2]

1.4.1 BFS STRATÉGIA

BFS stratégia spočíva v prechádzaní stromu po jeho urovňam: najprv začiatočný adresár (hĺbka 0), potom všetke vnorené adresári na hĺbke 1, potom všetke vnorené adresári na hĺbke 2, a tak ďalej. Štruktúra dat zodpovedá rádu. Uzly sa pridávajú na koniec, a vyberajú sa zo začiatku. Pomocou toho dosahuje sa úrovňoví poriadok prehliadania. [2]

Výhodami takého prehliadania je to, že čo najskôr dostaneme najbližšie k pracovnému adresáru súbory. Tak tiež taká stratégia umožňuje získavať potrebné súbory bez ponorenia do maximálnej hĺbke. Situačnou výhodou a nevýhodou takého riešenia je väčší náklad na pamäť v širokých stromoch, a menši v hlbokých.

1.4.2 DFS STRATÉGIA

DFS stratégia spočíva v prechode do maximálnej hĺbky prvej vetve stromu, návratu k začiatku stroma a ponorenia do ďalšej vetve. Štruktúra dát zodpovedá zásobníku, pri ktorom platí pravidlo „prvý sa vkladá, prvý sa vyberá“.

Výhodou je možnosť rýchlejšie dosiahnuť hlboké uzly. Situačne výhody a nevýhody sú opačne BFS stratégie – menši náklad na pamäť v širokých stromoch, a väčší v hlbokých.

1.4.3 REGEX AKO MECHANIZMUS VYHODNOTENIA

Pri spracovaní výsledkov vyhľadania adresára existuje možnosť zadania kritéria vyhodnotenia nie len priamym presným porovnaním. Je možne porovnanie zo šablónou, tak zvanými regulárnymi výrazmi (ďalej regex). Regex je formálny jazyk, ktorý umožňuje opísať viacero rôznych reťazcov s pomocou reťazca šablóny. [3]

V kontexte prechodu adresáre, potom, čo program dostane názov súboru alebo iný potrebný druh metá dát, skontroluje, či sa hodnota zhoduje so zadaným vzorom. Na rozdiel od priameho porovnania regex umožňuje vyjadriť všeobecnejšie kritérium, napríklad:

- Vyber všetkých súborov, názvy ktorých začínajú konkrétnou predponou;
- Vyber súborov, v ktorých sa v názve vyskytuje skupina znakov (napríklad dátum);
- Vyber z niekoľkých možností názvu prostredníctvom alternatív.

1.5 ČISTENIE PRIEČINKOV

Pod čistením priečinkov v súborovom systéme sa mysli odstránenie ne používateľských súborov, ktoré vznikajú, ako vedľajší produkt životného cyklu operačného systému, bežných programov alebo nástrojov pre vývoj softvéru. Sú to napríklad dočasne súbory, cache, logy, alebo havarijne výpisy. Ich spoločný príznak – oni nie sú prvotným výsledkom prace používateľa, ako napríklad uložený textový dokument, a nie sú nevyhnutnou súčasťou softvérových komponentov. Slúžia na zrýchlenie prace, zotavenia po nastavenie kritickej situácii alebo pre diagnostiku.

Vymazanie takých súborov sa berie, ako bezpečne primárne na základe jedného z dvoch princípov:

1. Dáta v súbore sú zotaviteľné – je možne dostať vymazané dáta opakovaným behom programu. „Štandard Hierarchie Súborového Systému“ priamo uvádza, že obsah priečinkov „cache“ sa vytvára v dôsledku drahých vstupov/výstupov alebo

výpočtov, a aplikácia ju musí byť schopná obnoviť a samotné súbory cache je možné vymazať bez straty údajov. [4]

2. Dáta sú dočasne – oni sú potrebne iba na krátke obdobie. „Štandard Hierarchie Súborového Systému“ priamo uvádza že programy nemajú rátať na to, že obsah priečinkov „tmp“ bude dostupný pri ďalšom behu. [4]

V ďalších podkapitolách sú uvedené typické kategórie súborov, ktoré môžu byť vymazané, dôvody prečo a teoretické možné negatívne dôsledky.

1.5.1 DOČASNÉ SÚBORY A ADRESÁRI

Predstavujú prechodné údaje vytvorené aplikáciami pri sťahovaní, rozbaľovaní, úprave, generovaní správ, inštalácii balíkov atď. Ich účelom je prežiť krátku fázu operácie. Odstránenie je bezpečne lebo po dokončení operácie už dočasný súbor zvyčajne nie je potrebný. Nevýhodou môže byť to, že ak taký súbor odstránite počas aktívnej operácie, môžete narušiť beh aplikácie. Príklady súborov a priečinkov: „*.tmp“, „*.temp“, „tmp/“, „temp/“.

1.5.2 SÚBORY NA ZOTAVENIE

Takéto súbory vytvárajú editori na ukladanie stavu aby zabezpečiť schopnosť zotaviť sa v prípade havárie. Sú bezpečne na vymazanie lebo zvyčajne nie sú potrebné po správnom zatvorení súboru. Nevýhodou môže byť to, že v prípade poškodenia hlavného súboru používateľ bude mat o dodatočnú zálohu menej. Príklady súborov: „*.swp“, „*.swo“.

1.5.3 REZERVNÉ AUTOMATICKÉ KÓPIE

Veľmi podobné na súbory na zotavenie, s tým rozdielom, že rátajú skôr nie s chybou aplikácie, ale používateľa, pre prípad, že potrebné zmeny nebudú uložene, napríklad prepisovanie a strate starej verzie súboru. Nevýhody odstránenia sú rovnaké, ako pri súboroch na zotavenie. Príklady súborov: „*.bak“, „*.old“, „*.autosave“.

1.5.4 LOG A DIAGNOSTICKÉ SÚBORY

Logy sú záznamy o činnosti aplikácie alebo systému. Diagnostické súbory vznikajú v prípade kritickej chyby v behu programu, aby uložiť vnútorný stav programu s cieľom diagnostiky, a nie uloženia používateľských dát. Síce oba druhý slúžia na diagnostické účely, oni môžu narasť do veľkých objemov. Odstránenie takých súborov žiadnym spôsobom neovplyvňuje beh programu. Iba v prípade, že máte technické problémy s

programom, odporúča sa nevymazávať také súbory kým problém nebude odhalení. Príklad súborov: „*.log“, „*.dmp“.

1.5.5 CACHE PRIEČINKY

Cache predstavujú neesenciálne dáta, ktoré urýchľujú opakovane spustenie procesov. S princípu ide o rekonštruovateľné dáta, ktoré sa ukladajú iba kvôli nákladom na ich vytvorenie. Často vznikajú vo vývojárskych programoch. A však vedia podobne, ako diagnostické dáta silno narastať v objeme a niekedy sú zdrojom chýb kvôli ponúkaniu neaktuálnych dát. Odstránenie obyčajne spomaľuje prvý ďalší beh procesu. Príklady priečinkov: „.cache/“, „.pytest_cache/“, „.ruff_cache/“.

1.6 SYSTÉM SPRAVY VERZII A GIT

Systém správy verzii (ďalej VCS) je systém, ktorý zaznamenáva zmeny súborov v čase tak, aby bolo možne dostať sa do konkrétneho stavu neskor. Vývoj softvéru není lineárny. Robia sa experimenty, chyby a návraty k starším riešeniam. VCS poskytuje históriu zmien, umožňuje porovnávať verzii, a prechádzať medzi stavmi alebo ich kombináciami bez manuálnych oprav kódu. [5]

Git je konkrétnym a najpopulárnejším VCS. Zakladanou jednotkou Git je repozitár. On predstavuje úložisko, v ktorom nachádzajú sa sledovane súbory a metadáta o ich zmenách. Zmena do repozitára ukladá, ako commit. Commit je snímkom stavu súborov v konkrétnom čase. Dôležitým je, že každý commit neobsahuje kopiju všetkých súborov, ale iba zmení v porovnanie s predošlým commitom. [5]

Síce Git funguje lokálne, ale na bezpečne uloženie a synchronizáciu prace v tíme sa bežné používa vzdialený repozitár. Jedným z najpopulárnejších hostingou pre vzdialene repozitári je GitHub.

1.7 PRÁCA Z OPERAČNOU PAMÄŤOU

Pri vývoje optimalizovaných projektov treba myslieť na špecifiku prace z operačnou pamäťou. Aj v prípadoch keď program není náročný na objem pamäte, alokácia alebo vydelenie pamäte pre dáta programu, môže byť časovo náročným procesom. Tak tiež prístup k dátam v operačnej pamäti môže mať rôznu rýchlosť. Zaleží to všetko hlavne od typu operačnej pamäte. Rozlišujeme zásobník (ďalej stack) a halda (ďalej heap) pamäť.

Stack funguje podľa pravidla „prvý sa vkladá, prvý sa vyberá“. Zásobník ukladá hodnoty v poradí, v akom ich získa, a odstraňuje hodnoty v opačnom poradí. Kvôli technickej realizácie stacku, doňho sa dostavajú iba dáta z fixovanou, známou v moment

kompilácie dĺžkou. Za tu to cenu dostávame časovo lacnejšiu alokáciu a prístup k dátam. [6]

Heap je menej organizovaným spôsobom odkladania do pamäte dat. Alokator vyhľadáva v pamäti dostatočne veľkí odsek pamäte z rezervou, pripadne ho výtvory reštrukturalizáciou pamäte, ukladá tam dáta a potom vkladá do stacku pointer na to kde sa nachádzajú v heape dáta. Je to výrazne pomalšie na alokáciu a mierne pomalšie na prístup. Ale zároveň umožňuje ukladať dáta veľkosť ktorých sa môže meniť alebo není známa v moment kompilácie. Dôležitou poznámkou je, že môžme mať viacero pointerov na rovnaké dáta, čim vieme silno ušetriť pamäť a množstvo alokácii.

2 CIELE PRÁCE

Hlavný cieľ bol stanovený, ako vývoj CLI nastroja pre Linux určený na organizáciu adresára, ktorý musí byť jednoduchý v použití, rýchlym a bezpečným. Prvé čo je potrebne špecifikovať v hlavnom ciele pojem „organizácia adresára“, zadefinovať konkrétne úlohy v obsluhe adresára, ktoré nastroj bude riešiť.

V bežnej prace z súborovým systémom hlavnými problémami sú nie úzko špecifikovane operácie, ale problémy z veľkým množstvom súborov a zložitou štruktúrou stromu adresára, v ktorých je ťažko sa zorientovať. Úlohou nastroja je minimálna uzavretá sada operácii nad takými adresármi: nájsť potrebné súbory, zoradiť adresár a vymazať nepotrebné nahromadené súbory. Taký vymedzený počet funkcii umožňuje zostať v rámci koncepcii komponentového systému, kde nastroj berie na sebe iba zodpovednosť za určitú vymedzenú oblasť prace.

Podmienka jednoduchosti sa spĺňa spomenutým vymedzeným počtom funkčných možnosti nastroja. K tomu treba dodať, že funkcionálne možnosti máju byť vyjadrené do minimálneho počtu jednoduchých príkazov z jednoduchou syntaxov, o ktorej používateľ ma sa vedieť dozvedieť bez dodatočných nástrojov a zdrojov.

Podmienka rýchlosti musí byť splnená aj pomocou použitia optimálnych algoritmov na prehliadanie adresára, aj rozumným použitím možnosti programovacieho jazyka pracovať z operačnou pamäťou. Síce pri prace z súborovým systémom nejde o veľké náklady na objem operačnej pamäte, ale časte a neopatrné vydelenie pamäte pre nastroj môžu ho výrazne spomaliť.

Potreba spoľahlivostí takého nástroja je odvodnená tým, že pri logickej chybe v kóde môže dojsť poškodeniu súborov a strate dát v nutrí nich. Preto treba zabezpečiť to, že kód sa sprava tak, ako je to popísane v jeho dokumentácii. Tak tiež z podmienky bezpečnosti vyplýva potrebnosť v možnosti sledovať prácu programu pre diagnostiku alebo havarijne prerušenie nastroja.

3 MATERIÁL A METODIKA

3.1 VÝBER TECHNOLÓGIÍ

Pred začiatkom vývoja treba určiť vhodné technológie ktorými bude implementované riešenie.

3.1.1 PROGRAMOVACÍ JAZYK

Najdôležitejšou technológiou je programovací jazyk. Musí umožniť realizáciu stanovených cieľov vrátané bezpečnosti a rýchlosti, a tak tiež mať zrozumiteľne náklady (predovšetkým časové) na vývoj nastroja. Pri výbere sme porovnali tri najrozšírenejšie programovacie jazyky pre vývoj CLI nástrojov na trhu: Python, C a Rust.

Python je najpopulárnejší jazyk na svete, a v jeho hlavné výhody patrí rýchlosť vývoja a jednoduchosť jazyka. Ale pri podrobenejšom pohľade sme zistili, že je to veľmi pomalý jazyk a programy na ňom máju nízku spoľahlivosť, aj keď zaviesť pravidla na explicitné určenie typov, ktoré nie su predvolené. To je v odpore zrazu z dvoma cieľmi, tak že sme zamietli tento programovací jazyk.

Ďalší na rade bol C. Klasický jazyk pre vývoj CLI nástrojov, a rodný jazyk pre Linux. Umožňuje písať najoptimálnejší možný kód, okrem možno priameho Asembler kódu, a ma dobrú ekosystému pre CLI aplikácie. Nevýhodou však stalo veľké časové náklady, a zároveň priama práca s operačnou pamäťou, čo môže byť dostatočne nebezpečne. Nevylúčili sme ten jazyk hneď, ale zamysleli sme sa či nebude posledný jazyk lepší.

Na konci zostal nám Rust. Najmladší zo všetkých jazykov, ponúkol, síce nižšiu než C, ale blízku k ním, rýchlosť. Zároveň, pri možnosti optimálnej práce s pamäťou, Rust je úplne pamäťovo bezpečný už na úrovní kompilátora. Pri tom, keďže v poslednom čase stáva sa veľmi populárnym na CLI nástroje, ma výborný ekosystém na tento účel. Takým spôsobom Rust bol zvolený, ako najoptimálnejší jazyk pre tento projekt.

3.1.2 VEDĽAJŠIE TECHNOLÓGIE

Stručne prejdeme vedľajšie technológie. Na prácu z file systémom bol zvolený „fs“ zo štandardnej craty, ako najlepšia a multiplatformová voľba. Pre zbieranie vstupných parametrov sme vybrali použil najpopulárnejšiu cratu v oblasti CLI nástrojov „clap“. Pre logovacie účely bol zvolený crate „log“, ktorý síce sa obyčajne používa sa vo väčších logovacích cratoch, ale vystačil pre naše účely. Ako VCS bol zvolený git.

3.2 ZALOŽENIE REPOZITÁRA A ZÁKLADNÁ ŠTRUKTÚRA

Po výbere technológii sme začali robiť základ pre projekt, ktorý bol by ľahko škálovatelný. Prvým krokom bolo založenie repozitára git a publikácia jeho vzdialenej verzii na platforme GitHub. Ďalej, po inicializácii nového projektu Rust cez cargo, pridali sme závislosti. Pre každú závislosť bola zvolená posledná stabilná verzia bez bezpečnostných rizík. Bezpečnosť bola overená cez cargo audit. Tak tiež závislosti sa pridávali z minimálnym možným množstvom voliteľných nastavení, aby finálna veľkosť nastroja nebola zbytočne veľká.

Základná štruktúra musí umožňovať ľahké pridané príkazov. Bolo rozhodnute, že nastroj bude mať minimálnu interaktivitu – keď je to možno, všetke vstupy sa budu zadávať, ako parametre spustenia nastroja, po čom nastroj overi vstupy, vykoná presne zadanú úlohu a ukonči svoju prácu. To zodpovedá klasickému správaniu CLI nástrojov – jedna úloha zodpovedá jednému spusteniu nastroja.

Na tom to základe vznikla štruktúra „Cli“, ktorá ukladala a validovala v sebe všetke vstupy a vytvárala potrebné globálne dáta. Aby štruktúra bola škalovatelnou, hlavným polom v nej sa stala trieda „Command“, ktorá obsahuje rôzne príkazy ktoré sa dajú postupne pridávať. Pritom každá varianta obsahuje metódu na spustenie, ktorá vždy príma „Cli“, ako jediný parameter.

3.3 VŠEOBECNÉ OPATRENIA

V priebehu vývoja nastroja viac krát sa opakovali rovnaké kroky alebo celkovo sa dodržovali konvencii stanovené na začiatku, preto v tejto kapitole budu pomenované všeobecne postupy, a to konkrétne: odchytávanie chyb, logovanie, optimalizácie a zabezpečenie bezpečnosti.

3.3.1 ODCHYTÁVANIE CHYB

Tento krok je súčasťou zabezpečenia bezpečnosti a spoľahlivosti nastroja. Väčšinu prace tu rieši samotná koncepcia Rust, ktorá núti nás priamo vyjadrovať sa ku každému momentu v ktorom operácia môže zlyhať. [6]

Priame vyjadrenie bolo zrealizované rovnako na všetkých miestach. Všetke chyby sa odchytávajú pomocou cratu „anyhow“, ktorý ukladá ich v sebe v podobe ne typovaných reťazcov. Keď chýba podľa názoru metódy není kritická, vykonajú sa podla potreby nejaké opatrenia a program pokračuje. Keď chyba je kritická, alebo metóda nevie o tom posúdiť, vráti chybu svojmu rodičovi, ktorý opakuje túto logiku, s tým, že môže

pridať chybe svoj kontext. Takým spôsobom kritická chyba vždy skončí v hlavnej metóde programu, ktorý ju zaloguje a bezpečne ukončí program.

3.3.2 LOGOVANIE

Každý objekt, ktorý sa priamo vypisuje, realizuje vlastnosť „Display“, ktorý je dostatočnou podmienkou pre logovanie cez štandardné výpisy do konzole. Logovanie v svojom rade je rozdelené na tri vrstvy, aby umožniť bezpečnostné opatrenia, medzi ktoré patrí možnosť sledovania akcii nastroja alebo diagnostika. Možne vrstvy, ktoré používateľ vie zapnúť parametrami nastroja:

- „Info“ – minimálne možná vrstva, ktorá je predvolanou;
- „Debug“ – v procese vykonania, nastroj bude informovať o tom aké akcie ide vykonať;
- „Trace“ – podrobný výpis informácii behu programu, vrátane vnútorného stavu nastroja.

3.3.3 OPTIMALIZÁCIE

Niektoré optimalizácie spočívajú v realizácie konkrétnych metód, ale v celom programe boli dodržané pravidla, ktoré zabezpečili väčšinu rýchlosti:

1. Všade kde je to možné, preferovať alokovanie pamäte na stack namiesto heap.
2. Nevytvárať nové kópie dát na heap, namiesto toho použiť referenciu na ne.
3. Všetke nepotrebné dáta máju byť v najskoršom čase uvoľnene.
4. Používať „lenivé“ iterácie namiesto priameho iterovania.

3.3.4 ZABEZPEČENIE SPOĽAHLIVOSTI

Hlavným zdrojom zraniteľnosti a havárií je samotný autor kódu, ktorý rátal s tým, že kód sa bude správať inak, alebo nerátal z nejakou vlastnosťou toho kódu.

Pre vylúčenie väčšiny takých prípadov pri vývoje nastroja boli napísane tak zváne modulové testy – testy na konkrétne metódy. Pre každú metóda, ktorá obsahuje v sebe logiku alebo algoritmus boli napísane testy. Samotné testy a kód potrebný pre nich je konfiguračné oddelený od bežného kódu, aby sa ne pridával do výsledného balíku pri finálnej kompilácie. Tak tiež testy sa ne pridávajú pre balíky kompilované pre platformu Windows. Kód priamo neobsahuje závislú od operačného systému logiku. Špecifické operácie sa nachádzajú v kóde testov, a zavádzanie multiplatformosti do nich malo by veľké náklady za tak mer žiadne zvýšenie spoľahlivosti kódu, keďže multiplatformovosť

kódu ktorý pracuje z súborovým systémom je otestovaná priamo v balíku ktorý na to bol použitý. Samozrejme bolo vykonané aj menej spoľahlivé manuálne testovanie nastroja. [7]

Okrem toho počas vývoja pomocou nastroja „clippy“ sa overovalo to, že žiadna chyba neprechádza priamo k operačnému systému, namiesto toho aby prejsť cez naprogramované odchytávanie chýb, čo garantuje správne ukončenia práce nastroja aj v prípade vyskytnutia chyb.

3.4 IMPLEMENTÁCIA HROMADNÝCH OPERÁCII

V tomto kroku sa pridala kľúčová logika pre nastroj – trieda, ktorá umožňuje vykonať akcie nad všetkými súbormi v adresáre - „FileList“.

Pri implementácie sa rozhodovalo medzi DFS a BSF stratégiou vyhľadávania, ktoré máju veľmi podobne vlasti a sú optimálnym spôsobom prechádzania všetkých súborov v priečinku. V konečnom dôsledku bola zvolená DFS stratégia, s dôvodu, že jedna z troch operácii pravdepodobne sa bude častejšie používať na priečinkoch, ktoré nemajú kritické pre stratégiu hniezdenie, keďže v praxi sortovanie nepoužíva sa na blízkom ku koreňu adresári. Pritom všetke tri operácie sa môžu často využívať v adresároch s veľkým množstvom prvkov, čo je neprijemne pre BSF. Dôležite však je nie konkrétna s tých dvoch stratégii, keďže reálny rozdiel je zanedbateľní, ale to, že sa použili pravé tieto najoptimálnejšie stratégie. [2]

Štruktúra bola optimalizovaná takým spôsobom, že ona vracia informácie iba o prvom súbore, a zapamätáva svoju pozíciu v priečinku. Štruktúra je určená na iterovanie, ktoré postupne bude vracať prvky, nad ktorými sa bude dat vykonať akcie. Takým spôsobom bude prejdene všetke prvky bez toho aby zbierať veľké množstvo dát.

Okrem toho štruktúra ma nastavenia na to, či ma rátať zo skrytými súbormi, a či vyhľadávanie ma byť rekurzívne. Druhé nastavenie umožňuje to aby sa na základe tejto štruktúry dalo spraviť prispôsobene BFS vyhľadávanie, čo bolo neskor použite.

3.5 IMPLEMENTÁCIA VYHĽADÁVANIA

S triedou „FileList“ väčšia časť vyhľadávania bola už spravená. Zostala iba spraviť mechanizmus na overovanie toho či sa pred nami nachádza súbor, ktorý hľadáme. Podmienky však musia byť ľahko škalovatelné a treba vyhnúť sa duplicite kódu. Preto bol vymyslený najkomplikovanejší v nástroje mechanizmus. Spravili sa tri vlastnosti:

- Extraktor – získava potrebnú hodnotu z metá dáta súboru
- Filter – určuje či porovnávame získanú hodnotu priamo alebo pomocou regexu
- FileMatcher – spája a kontroluje dva druhé vlasnosti

Keď tieto vlastnosti boli hotové, veľmi rýchlo bolo implementovane sa vyhľadávanie podľa mena, veľkosti, rozšírenia a dátumu vytvorenia súbora.

3.6 IMPLEMENTÁCIA ČISTENIA

Pri implementácie čistenia hlavnou úlohou bolo iba zadefinovať súbory a adresári, ktoré treba vymazávať a kvalitne otestovať kód, keďže to je veľmi citlivá operácia. Obe úlohy boli rýchlo dokončené.

3.7 IMPLEMENTÁCIA SORTOVANIA

Pri sortovaní prístup vyhľadania bol zmenení na BFS a pracou so zozbieraním informácii. Najprv sa zbierajú informácie o súboroch, potom sa sortujú do zoznamu originálnych a cieľových ciest, a len na konci sa presúvajú. Bolo to spravené z dôvodu toho, že pri presune rodičovského priečinku, mení sa aj cesta vnorených súborov. BFS umožnila medzi zmenou hĺbky obnovovať cestu na nové miesto. Zbieranie ciest umožnilo presun všetkých dát spolu, a sortovať vnútri už na novom mieste bez obavy neočakávaných zmien v moment prac nastroja. Implementované to bolo pomocou cyklov z vytvorením ne rekurzívnych tried „FileList“.

Boli implementované možnosti rekurzívneho a obyčajného sortovania, možnosť presunutia alebo kopírovania zdrojových dát a sortovania podľa rozsahu veľkostí, dátumu vytvorenia alebo rozšírenia (sortovanie podľa mená nema praktický význam).

3.8 BALÍK A INŠTALÁCIA

Po dokončení kódu, boli spravené dva finálne spustiteľné balíky – jeden pre Linux, druhý pre Windows. Tie to balíky boli nahrané na GitHub stránke projektu. Tak tiež bol spravený návod na inštaláciu, ako súčasť repozitára. Návod vysvetľuje, ako pridať nastroj do env a ako vytvoriť balíky zo zdrojového kódu.

3.9 PROPAGÁCIA A ÚDRŽBA

Tato kapitola opisuje navrhovaný postup toho, ako môže byt realizovaná údržba, rozvoj a dostupnosť nastroja pre používateľov.

3.9.1 PRÁVNY RÁMEC

Právnickým základom pre softvérový je jeho licencovanie. Vzhľadom na ďalej opísaný propagačný model, bola zvolená licencia MIT, ktorá patrí medzi permisívne licencie. Ona umožňuje používanie, úpravu a ďalšiu distribúciu, vrátane komerčných projektov. Pri distribúcie povinnosť je zachovanie licenčného textu a copyright informatici v kópiách použitého kódu. Pri tom my, ako autori projektu, ne nesieme žiadnu zodpovednosť za stav projektu. Je to jedná z najflexibilnejších licencií, a taký projekt sa klasifikuje, ako „open-source“. [8]

3.9.2 PROPAGÁCIA PROJEKTU

Základ propagácie tvorí ľahká dostupnosť a jasná prezentácia projektu. Repozitár musí byť verejným (open-source). Možnosť pozrieť si zdrojový kód projektu tak mer garantuje bezpečnosť projektu, lebo akýkoľvek škodlivý kód je rýchlo oddeliteľný a taký repozitár sa rýchlo blokuje. Pre dostupnosť bola vypracovaná integrovaná dokumentácia pre každý príkaz nástroja vo forme nového príkazu.

Repozitár musí obsahovať návod na inštaláciu projektu a už skompilovaný zo zdrojového kódu program. Skompilovaný program musí mať jasné označenie verzie a operačného systému. Aby verzovanie bolo konzistentným a pochopiteľným, použili sme sémantické verzovanie. Skladá sa s troch čísiel rozdelených bodkami. Prvá verzia signalizuje o nekompatibilných zmenách, druhá o pridaní novej funkcionality kompatibilným spôsobom, a tretia signalizuje o opravách chýb bez pridania nových možnosti. [9]

Podpora projektu je navrhovaná prostredníctvom „GitHub Issue“. Tento mechanizmus umožňuje používateľovi priamo v repozitáre informovať o chybe s podrobnými popisom chyby a žiadať o pridanie novej funkcionality. Rozvoj projektu, keďže on je „oupen-sorce“, sa spolieha na komunitu, ktorá vie prispieť kódom prostredníctvom „GithHub Pull Requests“, v ktorých autori projektu kontrolujú navrhnuté zmení a aplikujú ich. [10]

4 DISKUSIA

V danej prace bol navrhnutí a vyvinutí CLI nastroj pre organizáciu súborov. Keď, že CLI nástrojov existuje veľké množstvo, vrátane manažovanie súborového systému, porovnanie ich z vyvinutím nástrojom umožni hodnotenie jeho účelnosť.

Scenár prace zo súborovým systémom, rozdelení na vyhľadávanie súborov, ich triedenie a následne čistenie, v Linux sa najčastejšie výkonová pomocou nástrojov „find“, „mv“, „cp“, „rm“ a ich kombináciami. Kombinovanie takých nástrojov umožňuje pokrytie väčšini scenárov. Používateľ v podstate vytvára s príkazov svoj malý program. Nevýhodou toho však je ťažkosť použitia. Po prvé, pre vytváraní takých príkazov sú potrebné pokročilé vedomosti v prace z CLI nástrojmi. Po druhe, pri dlhých príkazoch je možne ľahko spraviť chybu, čo hlavne pri čistení má veľmi nepríjemné dopady. Filozofiou takých nástrojov v podstate je klasicky pre Linux prístup – za cenu ľahkosti použitia dostať čo najširšie možnosti ovládania systémom.

Naš nastroj na druhej strane ponúka veľmi jednoduchú syntax a použitie, keďže realizuje v sebe celý spomenutý cyklus práce s súborovým systémom. Cenou za to je pokrytie iba hlavných scenárov. Je to opačný prístup, kde ľahkosť použitia sa váži viac než možnosť vykonať ľubovolnú logiku. Síce je to menej typický pre Linux prístup, podľa nás hlavnou jeho filozofiou je možnosť prispôsobenia operačného systému svojim požiadavkám. Požiadavka jednoduchosti pripadá nám, ako realistická, hlavne pri prechodné na Linux z iných operačných systémov, kde CLI je ťažkou na osvojenie novinkou. Existujúcu do konca cele distribúcie, orientované na minimalizmus a jednoduchosť, ako napríklad Ubuntu a Mint. Na tomto základe si myslime že nastroj je účelný.

5 ZÁVERY PRÁCE

V rámci prace bol navrhnutí a vyvinutí CLI nastroj pre organizáciu súborov v adresári. Nastroj mal byť orientovaný na ľahkosť použitia, rýchlosť a bezpečnosť.

Jednou z hlavných úloh bola dekompozícia hlavného cieľu na vyplývajúce z praktických scenárov operácie na organizáciu súborov. V konečnom dôsledku boli implementované príkazy na 3 operácie – vyhľadávanie, sortovanie a inteligentné čistenie. Taký výber tvorí uzavretú sadu operácii, ktoré umožňujú organizáciu súborov ale neprerastajú do samostatného súborového manažéru, čo zodpovedá stanovenom cieľu.

Jednoduchosť použitia bola zrealizovaná pomocou spomenutou obmedzenou sadov operácií, jednoduchou syntaxov vysvetlenou integrovanou dokumentáciou a prehľadnou stránkou repozitára.

Podmienka rýchlosti bola zabezpečená optimalizačnými opatreniam, medzi ktoré pattry: vyber vhodného programovacieho jazyka, optimálne algoritmy prehliadania adresára a zmysluplný práca s pamäťou.

Bezpečnosť predovšetkým sa vnímala ako očakávané správanie programu. Preto všetke kľúčové miesta boli pokrytí modulovými testami. Pre kontrolu programu a jeho diagnostiku pre všetke príkazy boli zrealizované 3 úrovne logovania. Tak tiež, ako garancia toho, že v kóde sa nenachádza škodlivý kód, zdrojové súbory boli zverejnené na stránke repozitára. Bezpečnosť a spoľahlivosť všetkych použitych crate bola overená cez nastroj cargo audit.

ZOZNAM POUŽITEJ LITERATÚRY

[1] SHOTTS, William. The Linux Command Line. Fifth Internet Edition [online]. [S.l.]: LinuxCommand.org, 2019 [cit. 2026-02-18]. Dostupne na internete: <https://openlab.citytech.cuny.edu/emt2390l/files/2020/03/The-Linux-Command-Line-Book-5th-Edition.pdf>

[2] MORIN, Pat. Open Data Structures (in C++) [online]. [S.l.]: Open Data Structures, 2013 [cit. 2026-02-18]. Dostupne na internete: <https://opendatastructures.org/versions/edition-0.1c/ods-cpp.pdf>

[3] FREE SOFTWARE FOUNDATION, Inc. GNU Grep: Print lines that match patterns. Version 3.12, 2 January 2025 [online]. [S.l.]: Free Software Foundation, Inc., 2025 [cit. 2026-02-18]. Dostupne na internete: <https://www.gnu.org/s/grep/manual/grep.pdf>

[4] LSB Workgroup, The Linux Foundation. Filesystem Hierarchy Standard. Version 3.0 [online]. [S.l.]: The Linux Foundation, 2015-03-19 [cit. 2026-02-18]. Dostupne na internete: <https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf>

[5] CHACON, Scott a STRAUB, Ben. Pro Git. 2nd Edition [online]. [S.l.]: Apress, 2014 [cit. 2026-02-19]. Dostupne na internete: <https://git-scm.com/book/en/v2>

[6] KLÁBNIK, Steve a NICHOLS, Carol. The Rust Programming Language [online]. [S.l.]: Rust Project Developers, n.d. [cit. 2026-02-19]. Dostupne na internete: <https://doc.rust-lang.org/book/>

[7] THE RUST PROJECT DEVELOPERS. Module std::fs — Rust Standard Library [online]. [S.l.]: The Rust Project Developers, 2026-02-11 [cit. 2026-02-20]. Dostupne na internete: <https://doc.rust-lang.org/std/fs/>

[8] OPEN SOURCE INITIATIVE. The MIT License [online]. [S.l.]: Open Source Initiative, n.d. [cit. 2026-02-21]. Dostupne na internete: <https://opensource.org/license/mit>

[9] SEMVER.ORG. Semantic Versioning 2.0.0 [online]. [S.l.]: semver.org, n.d. [cit. 2026-02-21]. Dostupne na internete: <https://semver.org/>

[10] GITHUB, Inc. About issue and pull request templates [online]. [S.l.]: GitHub Docs, n.d. [cit. 2026-02-21]. Dostupne na internete: <https://docs.github.com/en/communities/using-templates-to-encourage-useful-issues-and-pull-requests/about-issue-and-pull-request-templates>škálovatelný