Ročníkový projekt (1) - zhrnutie vykonaných prác

Matej Fedor

February 7, 2019

Contents

1 Úvod

Tento dokument obsahuje rozbor prác vykonaných na ročníkovom projekte počas zimného semestra. Ich cieľom bolo vytvoriť základnú funkcionalitu užívateľskej konzolovej aplikácie pre operačné systémy na báze UNIXu. Všeobecnou úlohou aplikácie je vykonať validáciu zvolených systémových súborov na pamäťovom médiu kontrolovaného zariadenia a informovať používateľa o bezpečnostnej povahe týchto súborov. Cieľom je odhaliť nedovolenú a potenciálne škodlivú modifikáciu systémových súborov, ktoré mohol vykonať útočnník alebo malvér. Hlavnými charakteristikami aplikácie sú jej kompatibilita s čo možno najväčšou množinou UNIXových systémov, efektivita, kompaktnosť a intuitívne ovládanie. Práce na aplikácii sa v zimnom semestri nekončia, jej vývoj bude prebiehať aj naďalej, berúc do úvahy aktuálne potreby klienta. K vybudovaniu podobného softvéru je potrebných viacero zložiek, ktoré popíšeme bližšie v nasledujúcich častiach.

2 Štruktúra aplikácie

2.1 Základné koncepty

Východiskom našich úvah sa stáva nutnosť kompatibility aplikácie s čo najväčšou množinou systémov na báze UNIXu. Jej plánované použitie v praxi len zdôrazňuje, že je túto požiadavku potrebné brať vážne. Preto sme pri vývoji kládli dôraz na vhodný výber technológií, ktoré poskytujú okrem vyspelosti aj stabilitu a rozumnú spätnú kompatibilitu. Snažili sme sa minimalizovať množstvo závislostí na nutné minimum a vyhnúť sa zbytočnému používaniu neštandardných, ikeď možno pre programátora pohodlnejších, knižníc, ktoré by len rušili kompaktnosť našej aplikácie a mohli by skomplikovať jej použitie na niektorých systémoch. Práve naopak, striktne sme implementovali pomocou funkcionality garantovanej štandardom POSIX a štandardmi použitého programovacieho jazyka, ktorým je v našom prípade C++.

Všeobecnú úlohu popísanú v úvode bude aplikácia plniť nasledujúcim spôsobom. Každému zo zvolených súborov priradí jednoznačný identifikátor na základe ich obsahu a v špeciálne vytvorenej databáze sa pokúsi o tomto identifikátore, prípadne o názve súboru, získať ďalšie informácie. Spomínaná databáza vo svojej úplnej podobe nie je predmetom ročníkového projektu a na jej tvorbe a udržiavaní sa bude pracovať v neskorších fázach projektu.

Aplikáciu sme s ohľadom na jej prístup k úlohe rozdelili do niekoľkých nezávislých komponentov. Tento prístup nám umožnil okrem iného jednoduché pridávanie novej funkcionality bez toho aby bolo nutné meniť časti kódu, ktoré so zmenou priamo nesúvisia. Jej bloková štruktúra s abstraktnými rozhraniami zároveň dáva predpoklad pre celkovú kompaktnosť aplikácie aj napriek jej práci v rôznych odlišných režimoch. Pri vývoji sme teda zvážovali návrh komponentov pre vstup aplikácie, priradzovanie jednoznačného identifikátora súboru, komunikáciu s databázovým systémom a výstup aplikácie a paralelné spracovanie vstupu.

2.2 Vstup aplikácie

Kľúčovým vstupom aplikácie je samotné prehľadavanie súborového systému (Inupt::InputScanner). V súčasnosti trieda umožňuje rekurzívne prehľadávanie priestoru s definovaným koreňom v niektorom adresári alebo množine adresárov. Dobudúcna je ale plánované aj prehľadávanie na základe názvu súboru alebo množiny názvov a rovnako aj aplikácia filtrov pre prípony súborov.

Iným možným vstupom aplikácie je vstupný súbor (Input::InputFile). Takýto súbor obsahuje vo vhodnom formáte zoznam absolútnych ciest k súborom prehľadaného priestoru spolu s korešpondujúcimi jednoznačnými identifikátormi. Súbor je vytvorený priamo aplikáciou (2.4)

a umožňuje rozdeliť celkový proces validácie do viacerých krokov, respektíve vykonať jeho časť aj bez internetového pripojenia, kedy nie je možné komunikovať s databázou.

Pre metódy vstupu sme zvolili štandardné knižnice, ktoré nám zaručia širokú kompatibilitu naprieč rôznymi systémami. Pretože pri vstupe zo súboru už znova nepočítame jeho identifikátor, narozdiel od vstupu prehľadávaním, vzniklo mierne neelegantné porušenie zapúzdrenia metódy vstupu rozhraním Input a naša aplikácia sa musí v rôznych režimoch na základe metódy vstupu vetviť. Podobné principiálne problémy sa nám našťastie v ďalších komponentoch nevyskytli.

2.3 Jednoznačná identifikácia súboru

Pretože jednoznačnosť identifikátora je v našej aplikácii kľúčová, zvolili sme na jeho generovanie hashovaciu funkciu. Na konkrétnej funkcii nezáleží, v našej aplikácii vieme od konkrétnej implementácie abstrahovať (HashAlgorithm). Dôležité je, aby bola v aktuálnom čase považovaná za bezpečnú a nebolo možné v rozumnom čase nachádzať jej kolízie. Východiskom sa nám stal SHA256 (HashAlgorithm::SHA256), ktorý je dobrým kompromisom medzi bezpečnosťou identifikácie a miestom v pamäti, ktorý identifikátor zaberá. V budúcnosti ale plánujeme užívateľom umožniť aj využitie iných hashovacích funkcií, ktoré adresujú isté nedokonalosti SHA256.

So štandardnými knižnicami sme si tentokrát nevystačili a vzniká nám závislosť na knižnici Crypto++. Podporovaná je však na najrôznejších platformách od Redhat a Debian, cez Android, až po platformy Microsoftu. Problémy s kompatibilitou teda neočakávame, taktiež nám umožní jednoducho implementovať aj mnoho iných hashovacích funkcií.

2.4 Komunikácia s databázovým systémom a výstup aplikácie

Všetka práca vykonaná v doposiaľ spomenutých komponentoch sa zúročí v komponente Output. Primárna metóda získania výstupu aplikácie je založená na získaní potrebnej informácie z databázy (Output::OutputDBConnection). Veľký pozor bolo treba dať na samotný výber API, ktorým do databázy pristúpime. Tu totiž kladieme nároky nielen na systémy používajúce našu aplikáciu, ale aj na databázové servery, ktoré obsluhujú našu databázu. Z viacerých možností sme zvolili staršie C API založené na JDBC, ktoré nám umožní komunikáciu so sytémami MariaDB ako aj MySQL, a to s veľmi slušnou spätnou kompatibilitou a minimálnymi závislosťami na strane servera.

Parametrami pre vyhľadávanie v databáze sú v našej aplikácii identifikátor súboru a absolútna cesta k súboru. Na základe dát v databáze vieme používateľovi poskytnúť bližšie informácie o systémovom súbore, taktiež konštatovať, že je evidovaný, že je evidovaný ale na inom mieste v súborovom systéme, informovať užívateľa o existencii súboru s rovnakým názvom ale odlišným identifikátorom, prípadne konštatovať, že daný súbor nie je vôbec rozpoznaný. Kompletná informácia o skenovanom priestore je priebežne zapisovaná do výstupného súboru.

Výstup aplikácie môže byť aj súbor obsahujúci zoznam absolútnych ciest k súborom a ich identifikátorom (Output::OutputOffline). Takýto súbor je určený na neskoršie spracovanie aplikáciou, umožňuje rozdeliť spracovanie na viacero krokov, prípadne vykonať sken súborového priestoru aj bez sieťového pripojenia.

2.5 Paralelné spracovanie vstupu

Práce na tomto komponente ešte prebiehajú, sú v štádiu návrhu a čiastočnej implementácie. Čas ich zdieľania s verejnosťou nevieme odhadnúť. Motivácia riešenia je zjavná, akonáhle sa pokúsime spracovať vstupy s počtom súborov rádovo 10^5. Podľa našich meraní tento proces

zaberal už niekoľko minút a s rastúcou veľkosťou prehľadávaných súborov sa bude situácia dramaticky zhoršovať.

Pri návrhu metódy paralelného spracovania vstupu musíme zvážiť zložitosť vykonávania jednotlivých komponentov, ako aj to, že niektoré naše komponenty síce sú reentrant[1], no žiadny nie je thread-safe[2]. Input sa nám vo všeobecnosti paralelizuje veľmi zle. Čítanie zo vstupného súboru musí byť synchronizované, snaha paralelizovať prehľadávanie súborového systému efektívne by bola zbytočne zložitá a fungovala by len za určitých predpokladov. Samotné inkrementovanie iterátora InputScanner je naviac rýchle a nie je pre nás problémom. Preto budeme súborový priestor prehľadávať sekvenčne a vstupné informácie podávať jednotlivým vláknam aplikácie pomocou socketov. Pomocou socketov vieme uskotočniť obojstrannú komunikáciu a zaznamenávať si prípadné informácie o zlyhaní spracovania vstupov.

Podstatne väčší zmysel má o paralelnom riešení premýšľať v súvislosti s počítaním identifikátora súboru, ktorého časová zložitosť je závislá od veľkosti súboru. Databázový systém nám zasa zabezpečí bezpečný paralelný prístup do databázy, takže aj túto operáciu vieme vykonávať potenciálne efektívnejšie. Vykonávanie musíme synchronizovať pri zapisovaní do výstupných súborov, čo ale nie je problém, lebo vlákna v synchronizovanej časti kódu neostanú dlhú dobu.

Cieľom vznikajúcej triedy ParallelExecutor je, aby mal používateľ možnosť elegantne vykonávať spracovanie vstupu na ním určenom (rozumnom) počte jadier CPU.

3 Záver

V predošlom sme podrobne popísali koncepty, prístup k návrhu a vznik filozofie našej aplikácie. V súčasnosti neexistuje hotové konzolové rohranie pre používateľa aplikácie. To, na čom sme pracovali tento semester, je čo možno najkompaktnejší a najzrozumiteľnejší návrh kostry aplikácie a hierarchie tried komponentov, aby každá ďalšia úroveň aplikácie postavená na týchto komponentoch mohla byť prototypovaná rýchlo a elegantne. Tento prístup je pre nás veľmi dôležitý keďže podobnú aplikáciu plánujeme v budúcnosti implementovať aj pre platformy Microsoftu. Pre detaily súvisiace s implementáciou si, prosím, pozrite zdrojové kódy, ktoré sú k dispozícii na webovej stránke projektu. Komentáre spresňujúce zdrojový kód pripravujeme a pribudnú v najbližsom čase.

[1] komponenty možno vykonávať paralelne, ak každá paralelná vetva programu disponuje vlastnou inštanciou komponentu

[2] komponenty možno vykonávať paralelne, aj ak všetky paralelné vetvy programu zdieľajú jedinú inštanciu komponentu