Analiza projekta korišćenjem alata za verifikaciju softvera

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Tamara Đukić
tamarazdjukic@gmail.com

22. april 2024.

Sadržaj

1 Valgrind

Valgrind je profajler otvorenog koda koji nadgleda funkcionisanje programa i prijavljuje nepravilnosti u radu tog programa ukoliko one postoje.

Valgrind obuhvata sledeće alate:

- Memcheck (detektor memorijskih grešaka)
- Massif (praćenje rada dinamičke memorije)
- Callgrind (profajler funkcija)
- Cachegrind (profajler keš memorije)
- Hellgrind i DRD (detektori grešaka u radu sa nitima)

Instaliranje Valgrind-a: **sudo apt install valgrind**

1.1 Memcheck

Memcheck je najpoznatiji Vagrind-ov alat i on se podrazumevano poziva ako ne stavimo dodatnu opciju koji alat pozivamo. Memcheck detektuje memorijske greške korisničkog programa kao što su:

- Curenje memorije
- Neispravno oslobađanje memorije na hipu
- Čitanje ili pisanje u nedopuštenu memoriju na hipu, steku
- Korišćenje nedefinisanih vrednosti, vrednosti koje nisu inicijalizovane ili koje su izvedene od drugih nedefinisanih vrednosti

Pokrećemo skriptu **run_memcheck.sh** koja sadrži poziv memcheck-a. Rezultat izvršavanja se nalazi u **memcheck_result.txt**.

```
==13690==
==13690== LEAK SUMMARY:
==13690==    definitely lost: 19,960 bytes in 29 blocks
==13690==    indirectly lost: 191,046 bytes in 294 blocks
==13690==      possibly lost: 10,465,824 bytes in 87 blocks
==13690==    still reachable: 24,602,834 bytes in 62,488 blocks
==13690==                       of which reachable via heuristic:
==13690==                         multipleinheritance: 3,440 bytes in 10 blocks
==13690==         suppressed: 0 bytes in 0 blocks
==13690==
==13690== For lists of detected and suppressed errors, rerun with: -s
==13690== ERROR SUMMARY: 11110 errors from 73 contexts (suppressed: 0 from 0)
```

Slika 1: Rezultat memcheck-a

Zaključak: Primetimo da ima dosta memorije koja je definitivno i indirektno izgubljena. Takođe imamo dosta memorije kojoj možemo da pristupimo i da je oslobodimo.

1.2 Callgrind

Callgrind je alat koji u vidu grafa generiše listu poziva funkcija korisničkog programa. Podrazumevano, prikupljeni podaci se sastoje od:

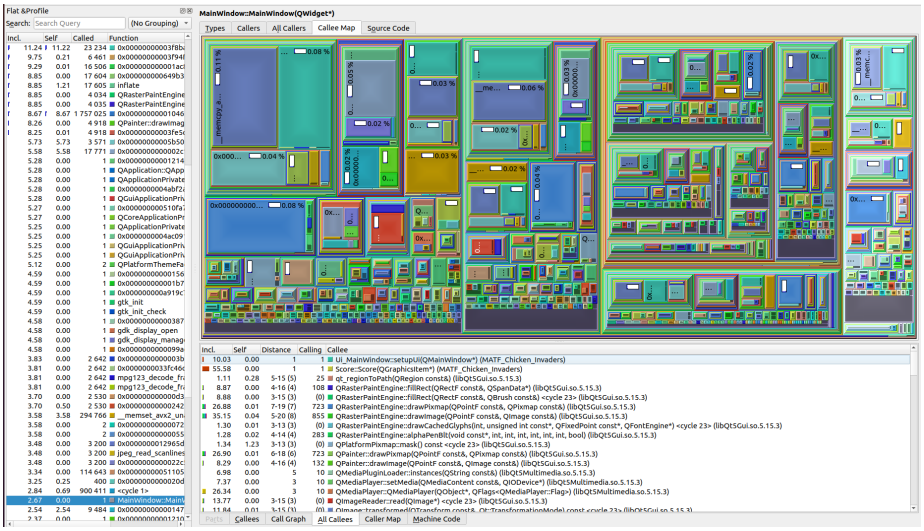
- Broja izvršenih instrukcija
- Njihovog odnosa prema izvornim linijama
- Odnos između pozivajućih i pozvanih funkcija
- Broj takvih poziva

Da bi alat Callgrind mogao da se pokrene, pre toga je projekat potrebno kompajlirati u profile režimu. Potrebno je da se kompajlira u ovom režimu da bi se izvršile dodatne optimizacije i da bi zapravo softver bio u stanju koje je slično onom za produkciju. Glavna razlika u odnosu na release režim je taj što u ovom režimu postoje dodatne debug informacije.

Pokrećemo preko skripte **run_callgrind.sh**.

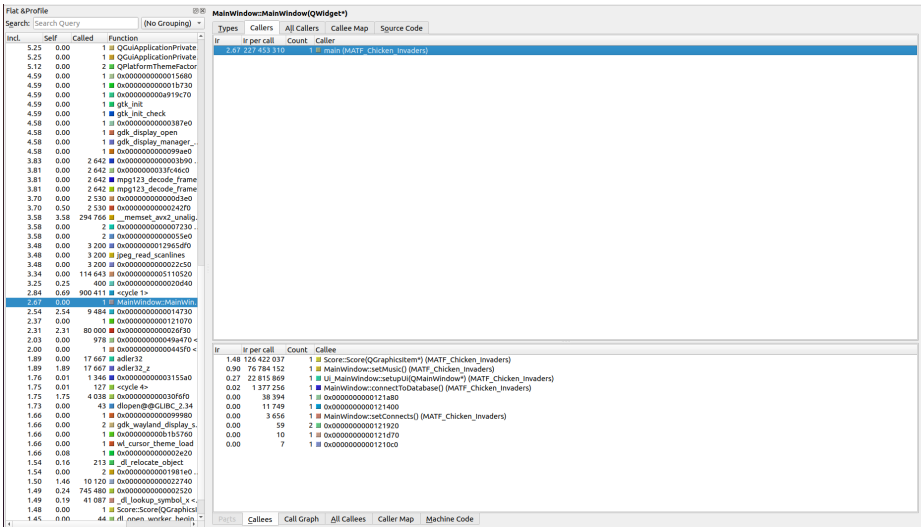
Kada se završi rad Callgrind-a, rezultati analize su zapisani u fajlu **callgrind.out.7326**. Otvorićemo taj fajl pomoću KCachegrind-a koji će nam grafički pokazati rezultate.

Prikaz Calle Map i All Calles za funkciju MainWindow(QWidget*):



Slika 2: Rezultat callgrind-a

Na levoj strani možemo videti broj pozivanja svake funkcije i broj instrukcija koje je zahtevalo njeno izvršavanje, samostalno i uključujući izvršavanja drugih funkcija koje je pozivala. Na desnoj strani smo izabrali opciju Callers koja prikazuje funkcije koje su pozivale funkciju MainWindow(QWidget*).



Slika 3: Rezultat callgrind-a

Zaključak: Posmatranjem izveštaja, može se zaključiti da nema velikog broja poziva funkcija u delu koda koji su implementirali programeri projekta.

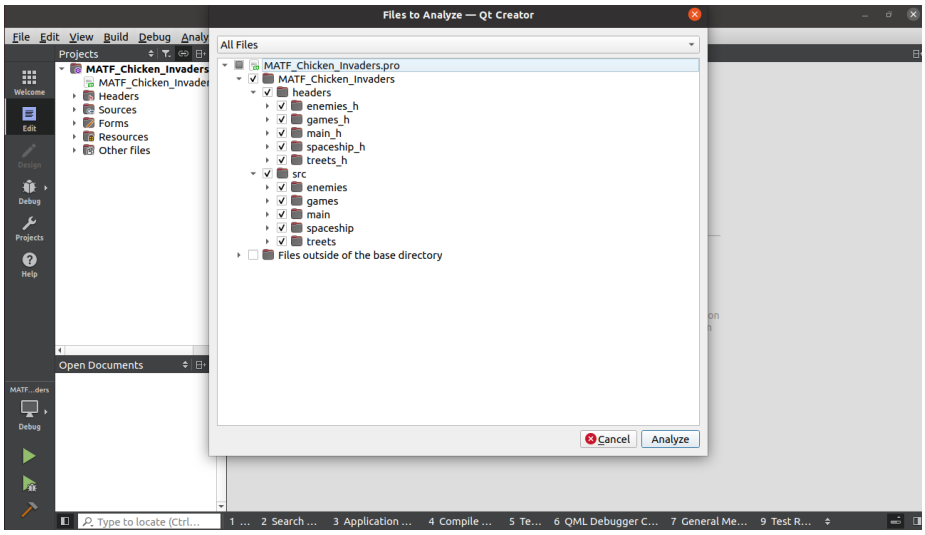
2 Clang

Clang je kompilator koji se koristi za jezike C, C++, Objective C i Objective C++. Posmatramo ga kao frontend koji na ulazu dobije kod koji je napisan u nekom od nabrojanih jezika i prevodi ga u međureprezentaciju. Backend vrši optimizacije vezane za konkretnu arhitekturu i da prevodi kod na mašinski jezik. Implementiran je u C++.

2.1 Clang-tidy

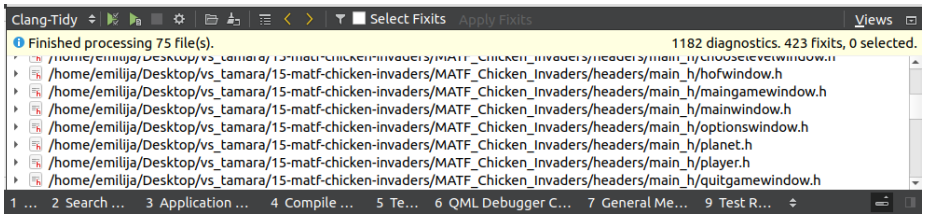
Clang-tidy statički analizator koda koji je zasnovan na Clang-u. Njegova uloga je da pomogne u prepoznavanju različiih programerskih grešaka kao što su kršenje stila pisanja, pogrešno korišćenje interfejsa ili da prepozna određene bagove koje je moguće pronaći bez prevođenja izvornog koda.

U okviru ovog projekta Clang-tidy je korišćen kao deo QtCreator-a. Da bi pokrenuli clang-tidy potrebno je kliknuti na tab Analyze i odabrati opciju Clang-tidy. Nakon toga, otvoriće se prozor u okviru kog možemo da biramo fajlove koje želimo da analiziramo:



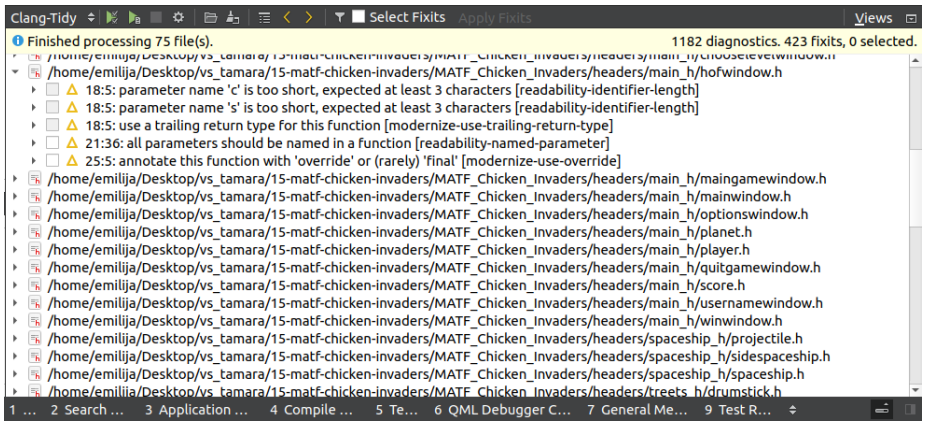
Slika 4: Odabir fajlova za clang-tidy analizu

Nakon pokretanja alata, i njegovog završetka, dobijamo sledeće:



Slika 5: Rezultat clang-tidy-a

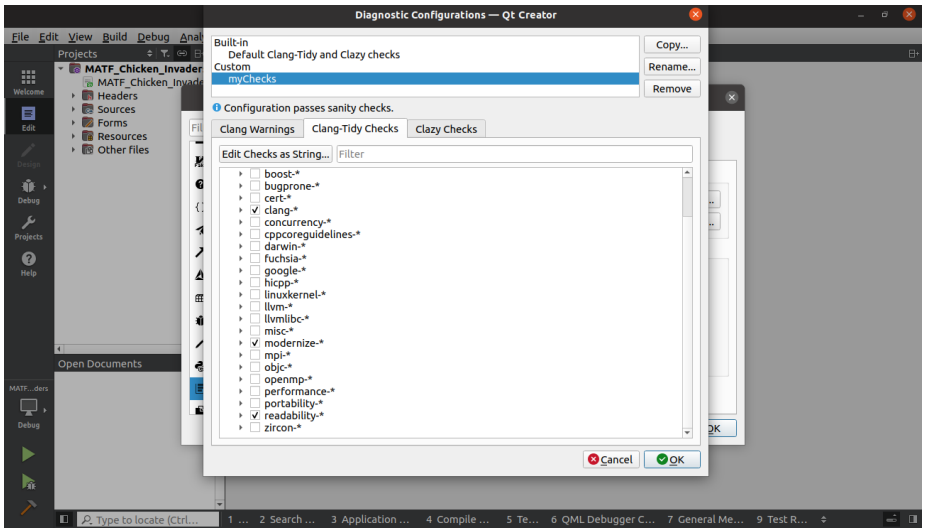
Ulazimo u neki fajl, i videćemo koje je greške našao:



Slika 6: Rezultat clang-tidy-a

Prilikom analize preko clang-tidy-a, možemo da biramo opcije koje će se

koristiti. Kliknemo na *Edit*, onda na *Preferences*, i izaberemo opciju *Analyzer*. Na slici biramo sami svoje opcije. Opcija **clang-*** je uvek uključena, dodali smo opcije **modernize*** i **readability-***.



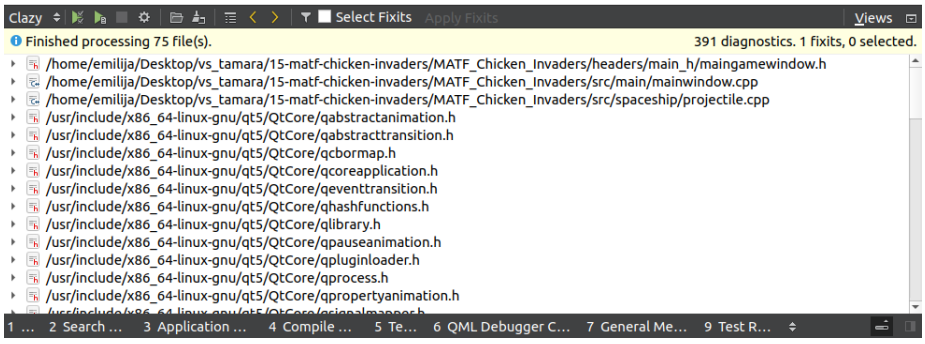
Slika 7: Biranje opcija za clang-tidy

Zaključak:

2.2 Clazy

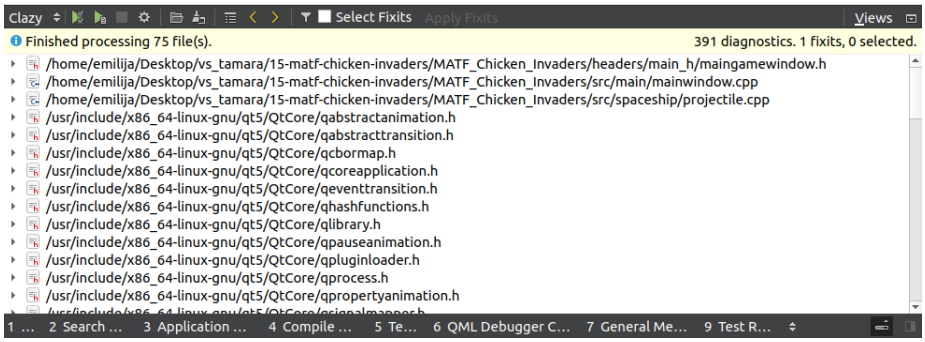
Clazy je alat koji pomaže Clang-u da razume semantiku Qt-a. Prikazuje upozorenja koja su povezana sa Qt-em koja mogu biti od nepotrebne alokacije memorije do toga da se API pogrešno koristi. Dodatno, može da prikaže akcije koje su potrebne da se srede neki od problema.

Da bi pokrenuli clang-tidy potrebno je kliknuti na tab Analyze i odabrati opciju Clang-tidy. Nakon toga, otvoriće se prozor u okviru kog možemo da biramo fajlove koje želimo da analiziramo. Nakon završetka analize, dobijamo sledeće:



Slika 8: Rezultat clazy-a

Ako uđemo u neki fajl, videćemo koje je greške našao:



Slika 9: Rezultat clazy-a

Zaključak:

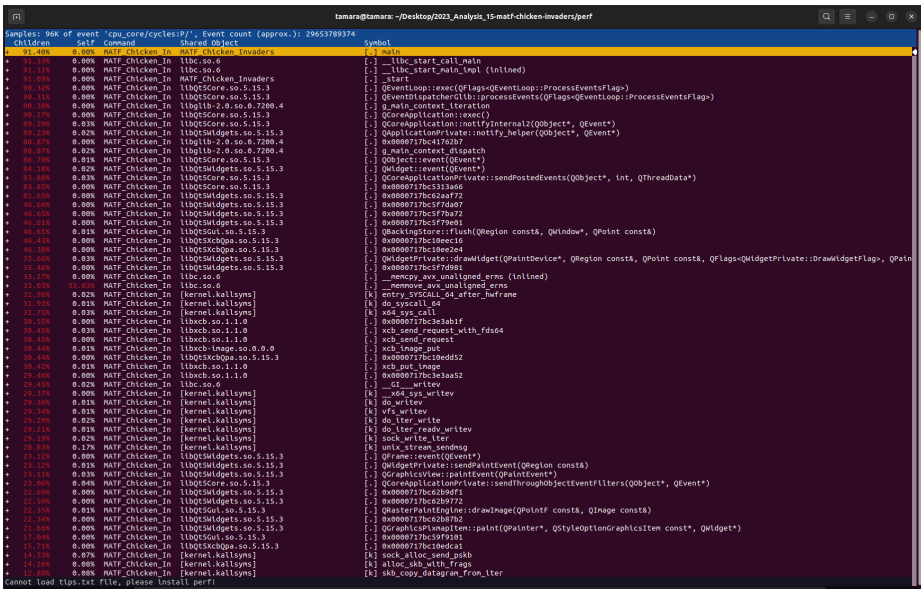
3 Perf

Perf je alat za profajliranje na Linux sistemima. U zavisnosti od opcija koje mu se navedu Perf može da prati različite događaje, da se nakači na određeni proces kao i da pravi uzorke odnosno profile na nivou niti, procesa ili procesora.

Da bi se instalirao ovaj alat potrebno je pokrenuti narednu komandu koja će u zavisnosti od kernela skinuti odgovarajuću verziju: **sudo apt-get install linux-tools-$(uname -r)**.

Za pokretanje alata Perf koristimo skriptu **run_perf.sh**. Argument –call-graph označava da Perf prikuplja informacije o grafu poziva funkcija, a argumentom dwarf označavamo da želimo da prikupimo informacije o steku.

Da bi se rezultat perf-a prikazao bolje, pozivamo komandu **sudo perf report**.



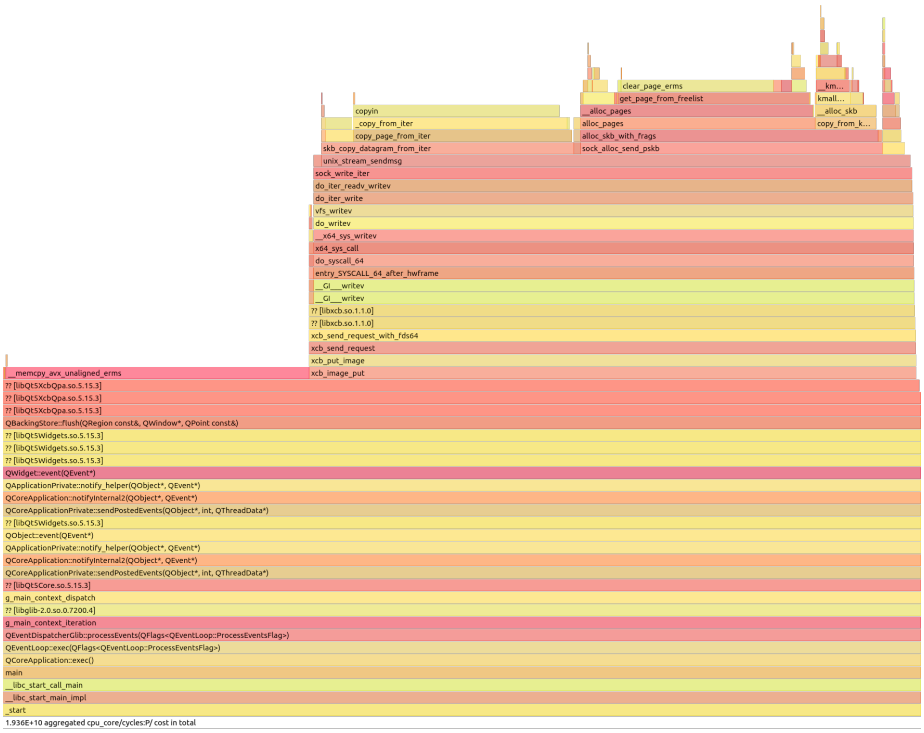
Slika 10: Rezultat perf-a

U koloni Children se nalazi koliko vremena se izvršava funkcija zajedno sa funkcijama koje ona poziva u odnosu na ukupno vreme izvršavanja programa izraženo u procentima. Kolona Self ozačava koliko vremena se sama funkcija izvršava u odnosu na ukupno vreme izvršavanja programa izraženo u procentima.

Iz ovog izveštaja možemo videti da najviše vremena se izvršavaju funkcije iz Qt biblioteke, takođe neke kerelove funkcije.

Za vizuelizaciju pomoću vatrenog grafika može da se koristi alat hotspot, projekat otvorenog koda. Hotspot je alat koji služi za vizuelizaciju izveštaja dobijenih od Perf alata, a ujedno je moguće pokrenuti i perf analizu iz grafičkog korisničkog interfejsa.

Na x-osi je populacija uzorka, dok y-osa predstavlja dubinu steka.



Slika 11: Rezultat hotspot-a

Zaključak: