

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	2
1.1	Memcheck	2
1.2	Callgrind	3
2	Clang	4
2.1	Clang-tidy	4
3	Perf	4

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 Valgrind-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**.

```
1 ==13690==
2 ==13690== LEAK SUMMARY:
3 ==13690==    definitely lost: 19,960 bytes in 29 blocks
4 ==13690==    indirectly lost: 191,046 bytes in 294 blocks
5 ==13690==    possibly lost: 10,465,824 bytes in 87 blocks
6 ==13690==    still reachable: 24,602,834 bytes in 62,488 blocks
7 ==13690==                of which reachable via heuristic:
8 ==13690==                    multipleinheritance: 3,440 bytes in 10 blocks
9 ==13690==    suppressed: 0 bytes in 0 blocks
10 ==13690==
11 ==13690== For lists of detected and suppressed errors, rerun with: -s
12 ==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.

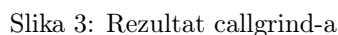
Prikaz Calle Map i All Calles za funkciju MainWindow(QWidget*):



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*).

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



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++.

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

3 Perf

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)**.

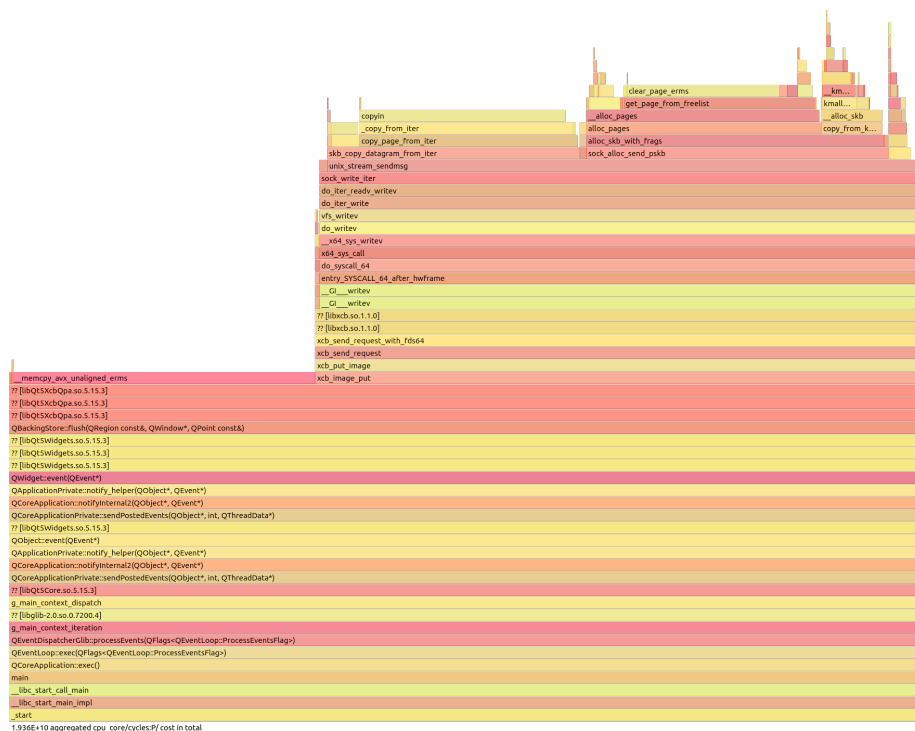
4

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



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

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



Slika 5: Rezultat hotspot-a