Analysis Moxit-Mock Dixit

Praktični seminarski rad u okviru kursa Verifikacija softvera
Univerzitet u Beogradu, Matematički faklutet

Andrijana Bosiljčić

26. januar 2024.

Sažetak

U ovom dokumentu će biti prikazan izveštaj analize studentskog projekta "Moxit-Mock Dixit", koja je dobijena primenom alata za verifikaciju softvera. Projekat je rađen na kursu "Razvoj softvera", u C++ programskom jeziku i Qt radnom okviru.

Sadržaj

1 Uvod

Moxit je implementacija društvene igre Dixit u C++. Narator se smenjuje među igračima po rundama u krug. Narator bira kartu i vezuje asocijaciju u vidu reči ili rečenice za odabranu kartu. Bez otkrivanja naratorove karte, ali sa otkrivanjem njegove asocijacije, igrači svaki za sebe biraju kartu koja ga najviše asocira na datu asocijaciju. Potom sledi kolektivno glasanje gde igrači koji nisu narator pogađaju naratorovu kartu. Poeni se dodeljuju u skladu sa time da li je karta pravilno pogođena - i ako nije - čija karta je pogrešno pogođena. Igračima je cilj da što pre stignu do skupljenih 30 bodova. Prvi koji stigne do kraja je pobednik. Frontend i backend su aplikacije za sebe i svaka zahteva izgradnju i pokretanje zasebno. U skladu sa time je i izvršena analiza čitavog projekta.

2 Statička analiza koda

Za statičku analizu koda korišćena su sledeća dva alata: Clang-Tidy i Cppcheck. Oba alata analiziraju kod bez njegovog izvršavanja, sa ciljem da se poboljša kvalitet koda.

2.1 Clang-Tidy

Clang-Tidy je alat baziran na Clang kompajleru, koji analizira izvorni kod programa, detektuje potencijalne probleme i daje predloge izmena u kodu, koji mogu rešiti te probleme. Neki od tih problema su bagovi, bezbednosni propusti, stilski problemi. Clang-Tidy može automatski da reši neke probleme, poput stilskih i uobičajenih programerskih grešaka. Takođe, može automatski da modernizuje kod. To se može postići navođenjem neke od opcija $--$**fix-errors** ili $--$**fix** prilikom pokretanja ovog alata.

Clang-Tidy je za potrebe statičke analize koda ovog projekta pokrenut nad svim *.cpp* i *.h* fajlovima, korišćenjem alata **run-clang-tidy**. Rezultati pokretanja ovog alata mogu se naći u fajlovima **output_backend.txt** i *output_frontend.txt*.

Na frontend delu aplikacije nisu uočena nikakva upozorenja, osim par propusta vezanih za **fmt** biblioteku, koja je već gotova uključena u projekat. Propusti su vezani za polja klase koji nisu inicijalizovani u okviru konstruktora.

Na backend delu aplikacije uočeno je potencijalno curenje memorije u klasi MoxitServer u 151. i 381. liniji koda. U funkcijama **addEpollWriteEvent** i **acceptConnection** dinamički je alocirana memorija operatorom **new** i ta memorija nije oslobođena. Ovaj propust se može ispraviti tako što se na kraju obe funkcije, pre **return** naredbe, oslobodi memorija korišćenjem operatora **delete**.

2.2 Cppcheck

Cppcheck je alat koji analizira izvorni kod programa, kako bi se identifikovali potencijalni problemi, kao što su curenje memorije, prekoračenje

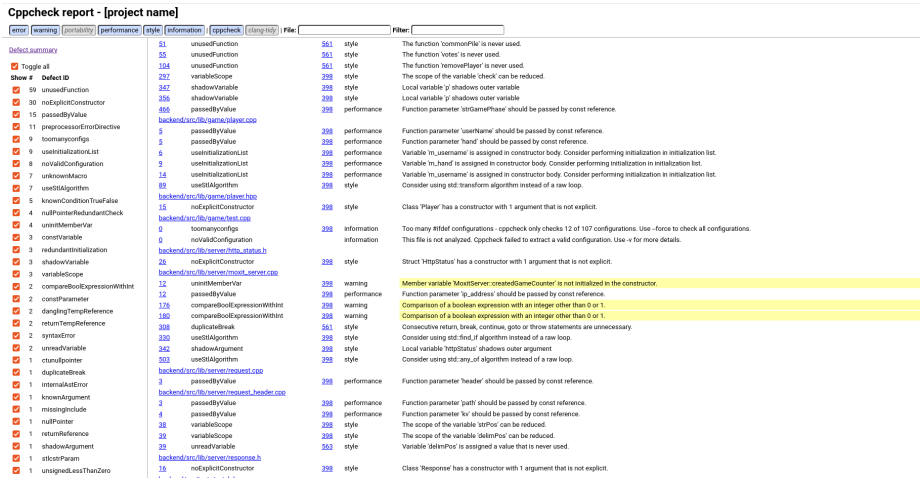
bafera, neinicijalizovane promenljive, problemi sa pokazivačima.

Analiza koda alatom Cppcheck je pokrenuta skriptom **cppcheck.sh**. Pokretanjem ovog skripta generiše se izveštaj u kome su preglednije predstavljeni propusti koje je uočio ovaj alat 1. Izveštaj se nalazi u **report/index.html**.

Što se tiče stilskih propusta u ovom projektu, oni se tiču funkcija i promenljivih koji nisu korišćeni, konstruktora koji primaju jedan argument, a nisu deklarisani ključnom rečju **explicit**, unutrašnjih promenljivih koje sakrivaju spoljašnje promenljive, redundantno inicijalizovanih vrednosti promenljivih.

Upozorenja na koja je alat Cppcheck ukazao tiču se poređenja boolean promenljivih sa **true** ili **false**, umesto sa **0** ili **1**, neinicijalizovanih polja klase u konstruktoru te klase.

Cppcheck je dao i preporuke korišćenja algoritama standardne biblioteke, umesto manuelnog iteriranja kroz petlju, deklarisanja promenljivih ključnom rečju **const**. Takođe je preporučio da se parametri funkcija prosleđuju po const referenci, a ne po vrednosti. To je korisno u situacijama kada funkciji prosleđujemo veliki objekat. Na taj način štedimo memoriju i poboljšavamo performanse programa.



Slika 1: Deo izveštaja generisanog na osnovu rezultata dobijenih pokretanjem Cppcheck alata

3 Profajliranje

Profajliranje je tehnika dinamičke verifikacije softvera. Alati za profajliranje tokom izvršavanja beleže podatke o programu i zatim prave njegov profil, na osnovu kog se mogu pronaći uska grla programa. Za profajliranje ovog projekta korišćeni su *Valgrind-ovi* alati i alat *Perf*.

3.1 Valgrind

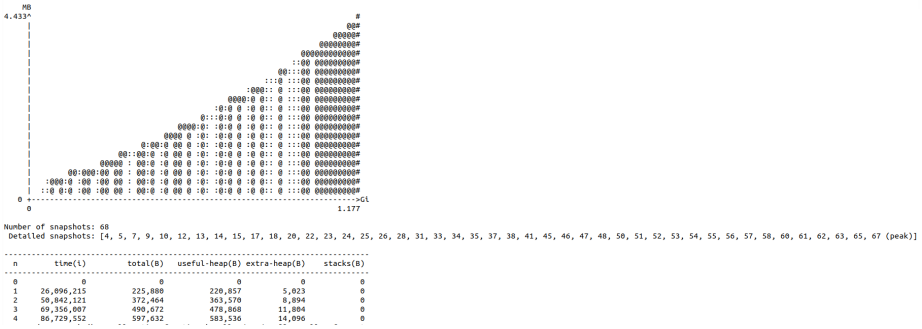
Valgrind distribucija nudi mnoge korisne alate za profajliranje koji omogućuju da se detektuje curenje, velika potrošnja memorije, greške u višenitnim programima, upotreba keš memorije. Valgrind-ovi alati korišćeni za profajliranje ovog projekta su *Massif* i *Cachegrind*.

3.1.1 Massif

Massif je alat koji meri koliko heap memorije koristi program i pruža informacije o tome koji delovi programa su odgovorni za alokaciju memorije. Takođe može da identifikuje curenje memorije koje se dešava u situacijama kada u programu postoji pokazivač na memoriju koja se više ne koristi. Ovaj alat omogućava i merenje zauzeća memorije na steku, što se postiže uključivanjem dodatne opcije **−−stacks=yes**, koja dosta usporava rad alata.

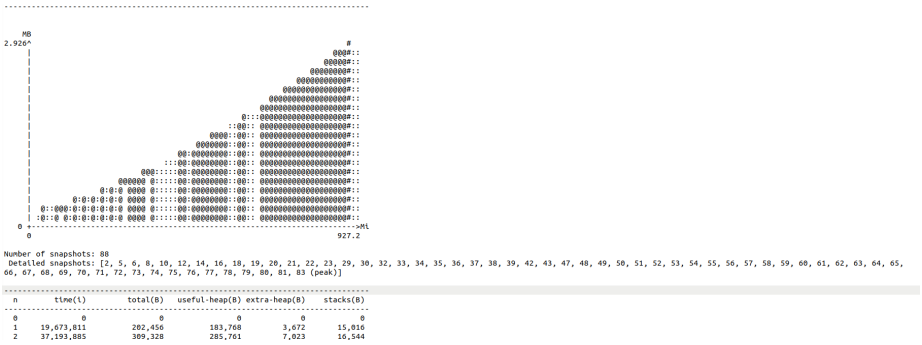
Massif je na ovom projektu pokrenut zasebno za backend i frontend deo aplikacije, korišćenjem sledećih skriptova: **massif_backend.sh** i **massif_frontend.sh**. Izveštaji ovog alata upisani su u fajlove **massif.out.<pid>**, gde je **<pid>** ID procesa. Korišćenjem skripta **ms_print.sh**, dobijaju se čitljivije informacije iz izveštaja, koje su zapisane u fajlovima **massif_<pid>.txt**. Na slikama 2, 3, 5, 4 se mogu videti te informacije u vidu grafa, gde je na x osi predstavljeno vreme (podrazumevano izraženo brojem izvršenih instrukcija), a na y osi je predstavljena ukupna količina memorije na hipu koju program zauzima u datom vremenskom trenutnku.

Na backend delu aplikacije Massif je napravio 68 preseka stanja, dok je na frontend delu aplikacije napravio 54 preseka stanja. Pik (eng. *Peak*) je najveća tačka potrošnje memorije koju je Massif zabeležio prilikom profilisanja programa. Na backend delu aplikacije pik je dostignut u 67. preseku i iznosi 4.433MB, a na frontend delu pik je dostignut u 51. preseku i iznosi 44.31MB. Poredeći preseke koje je Massif zabeležio, i na backend i na frontend delu aplikacije može se zaključiti da potrošnja memorije postepeno raste i da je najveća u preseku 67 na backend delu aplikacije, a na frontend delu u preseku 51. Takođe se može primetiti da se nova memorija alocirala od drugog preseka. U izveštaju su takođe prikazani i podaci o određenim presecima, gde se može videti utrošak korisno alocirane memorije, extra memorije koja je morala biti alocirana, kao i količinu memorije u bajtovima koju koristi stek programa (ukoliko je prilikom pokretanja Massif alata bila uključena opcija **−−stacks=yes**). Na slikama 2, 3 nalazi se jedan od izveštaja Massif alata pokrenutog na backend delu aplikacije sa i bez uključene opcije **−−stacks=yes**, dok se na slikama 5, 4 može videti isto, samo na frontend delu aplikacije.



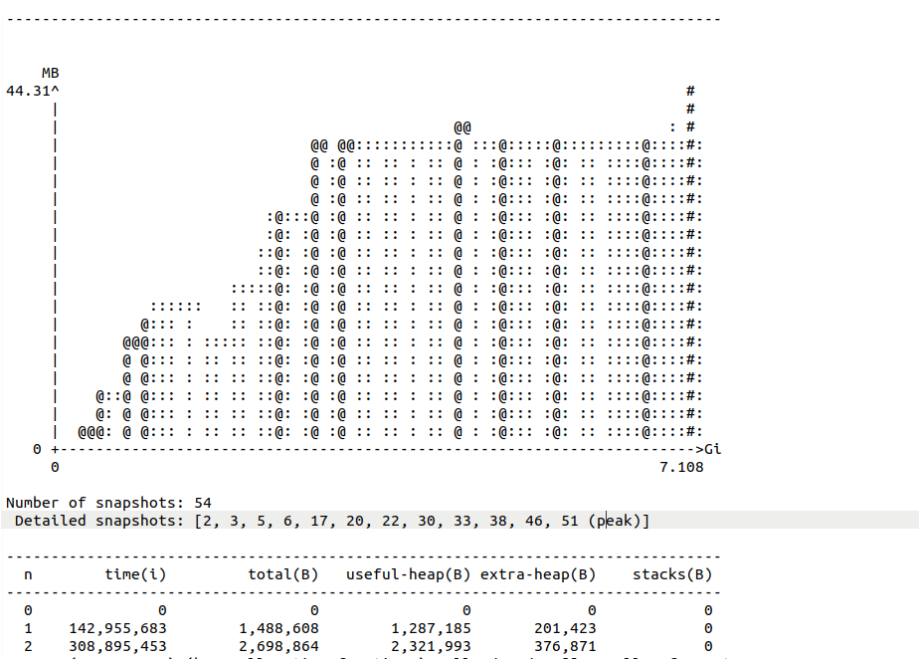
```
    MB
4.433^                                                                  #
     |                                                                  @@#
     |                                                                @@@@#
     |                                                              @@@@@@#
     |                                                            @@@@@@@@#
     |                                                          @@@@@@@@@@#
     |                                                        ::@@ @@@@@@@@#
     |                                                      @@:::@@ @@@@@@@@#
     |                                                    :::@ :::@@ @@@@@@@@#
     |                                                 :@@@:: @ :::@@ @@@@@@@@#
     |                                               @@@@:@ @:: @ :::@@ @@@@@@@@#
     |                                             :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |                                          @:::@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |                                       @@@@:@: :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |                                  @@@@ @ :@: :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |                             @:@@:@ @@ @ :@: :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |                        @@::@@:@ :@ @@ @ :@: :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |                   @@@@ : @@:@ :@ @@ @ :@: :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |          @@:@@@:@@ @@ : @@:@ :@ @@ @ :@: :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |   :@@@:@ :@@ :@@ @@ : @@:@ :@ @@ @ :@: :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
     |  ::@ @:@ :@@ :@@ @@ : @@:@ :@ @@ @ :@: :@:@ @ :@ @:: @ :::@@ @@@@@@@@#
   0 +-------------------------------------------------------------------------->Gi
     0                                                                   1.177

Number of snapshots: 68
 Detailed snapshots: [4, 5, 7, 9, 10, 12, 13, 14, 15, 17, 18, 20, 22, 23, 24, 25, 26, 28, 31, 33, 34, 35, 37, 38, 41, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 60, 61, 62, 63, 65, 67 (peak)]

--------------------------------------------------------------------------------
  n        time(i)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
  0              0                0              0              0            0
  1     26,096,215          225,880        220,857          5,023            0
  2     50,842,121          372,464        363,570          8,894            0
  3     69,356,007          490,672        478,868         11,804            0
  4     86,729,552          597,632        583,536         14,096            0
```

Slika 2: Deo izveštaja Massif alata pokrenutog na backend delu aplikacije bez uključene opcije $--$**stacks=yes**

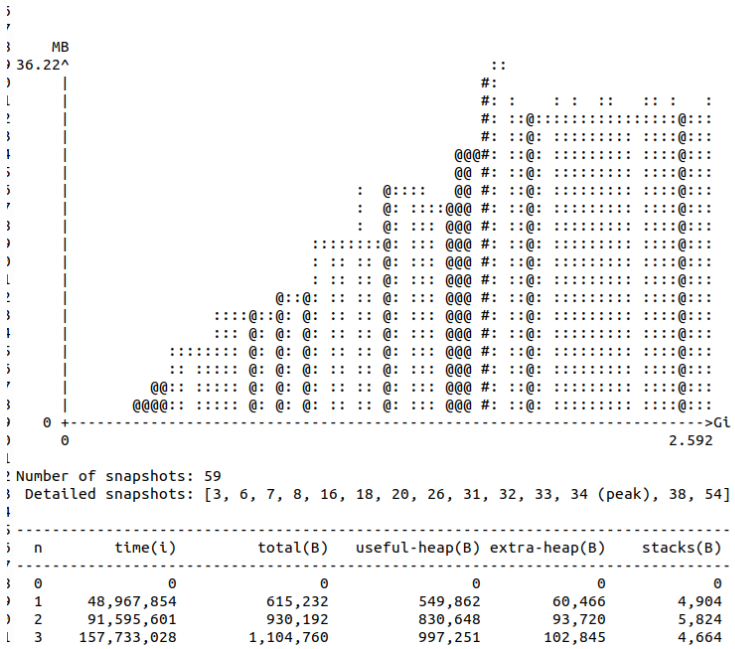


```
--------------------------------------------------------------------------------
    MB
2.926^                                                                  #
     |                                                                @@@#::
     |                                                              @@@@@#::
     |                                                            @@@@@@@#::
     |                                                          @@@@@@@@@#::
     |                                                        @@@@@@@@@@@@#::
     |                                                      @@@@@@@@@@@@@@#::
     |                                                    @@@@@@@@@@@@@@@@#::
     |                                                  @::@@@@@@@@@@@@@@@@#::
     |                                               ::@@:: @@@@@@@@@@@@@@@@#::
     |                                             @@@@::@@:: @@@@@@@@@@@@@@@@#::
     |                                          @@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
     |                                       @@:@@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
     |                                  :::@@:@@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
     |                             @@@:::::@@:@@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
     |                        @@@@@@ @:::::@@:@@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
     |                   @:@:@ @@@@ @:::::@@:@@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
     |          @:@:@:@:@:@ @@@@ @:::::@@:@@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
     |  @::@@@:@:@:@:@:@:@ @@@@ @:::::@@:@@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
     | :@::@ @:@:@:@:@:@:@ @@@@ @:::::@@:@@@@@@@::@@:: @@@@@@@@@@@@@@@@#::
   0 +-------------------------------------------------------------------------->Mi
     0                                                                   927.2

Number of snapshots: 88
 Detailed snapshots: [2, 5, 6, 8, 10, 12, 14, 16, 18, 19, 20, 21, 22, 23, 29, 30, 32, 33, 34, 35, 36, 37, 38, 39, 42, 43, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 83 (peak)]

--------------------------------------------------------------------------------
  n        time(i)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
  0              0                0              0              0            0
  1     19,673,811          202,456        183,768          3,672       15,016
  2     37,193,885          309,328        285,761          7,023       16,544
```

Slika 3: Deo izveštaja Massif alata pokrenutog na backend delu aplikacije sa uključenom opcijom $--$**stacks=yes**



Slika 4: Deo izveštaja Massif alata pokrenutog na frontend delu aplikacije bez uključene opcije $--$**stacks=yes**



n	time(i)	total(B)	useful-heap(B)	extra-heap(B)	stacks(B)
0	0	0	0	0	0
1	48,967,854	615,232	549,862	60,466	4,904
2	91,595,601	930,192	830,648	93,720	5,824
3	157,733,028	1,104,760	997,251	102,845	4,664

Slika 5: Deo izveštaja Massif alata pokrenutog na frontend delu aplikacije sa uključenom opcijom $--$**stacks=yes**

3.1.2 Cachegrind

Cachegrind je alat koji omogućava profajliranje keš memorije. Prati pristupe memoriji, pogotke i promašaje u keš memoriji i statistiku korišćenja keš memorije. Postoje mnogi alati za vizuelizaciju izlaza alata Cachegrind. Jedan od njih je KCachegrind, koji pruža interaktivni interfejs za istraživanje podataka profilisanja i grafikona poziva.

Alat Cachegrind je na ovom projektu pokrenut zasebno za backend i frontend deo aplikacije, korišćenjem sledećih skriptova: **cachegrind_backend.sh** i **cachegrind_frontend.sh**. Prilikom pokretanja alata uključene su sledeće opcije: $--$**keep**$-$**debuginfo=yes**, koja nalaže ovom alatu da zadrži informacije o otklanjanju grešaka(simbolička imena za promenljive, imena funkcija i brojeve redova izvornog koda) dok simulira ponašanje keša, i opcija $--$**cache**$-$**sim=yes**, koja omogućava simulaciju keš memorije. Izveštaji ovog alata upisani su u fajlove **cachegrind.out.<pid>**, gde je <**pid**> ID procesa. Korišćenjem skripta **cg_annotate.sh**, dobijaju se čitljivije informacije iz izveštaja, koje su zapisane u fajlovima **cachegrind_<pid>.txt**, a ukoliko se prlikom pokretanja ovog skripta uključi opcija $-k$ pokreće se alat KCachegrind koji vizualizuje i analizira podatke o profilisanju generisanje od strane alata Cachegrind. Na slici 6 se mogu videti informacije o keš memoriji koje se nalaze na početku izveštaja. I1 i D1 redom predstavljaju sekciju keš memorije u koju se smeštaju instrukcije i sekciju keš memorije u koju se smeštaju podaci, a zajedno pripadaju prvom nivou keš memorije, dok je LL poslednji nivo keš memorije koji je objedinjen. Kod ovih oznaka prvi broj predstavlja veličinu keš memorije izraženu u bajtovima, drugi broj predstavlja veličinu keš linije izraženu u bajtovima, a treći broj predstavlja keš asocijativnost.

```
I1 cache:         65536 B, 64 B, 2-way associative
D1 cache:         65536 B, 64 B, 2-way associative
LL cache:         262144 B, 64 B, 8-way associative
```

Slika 6: Cachegrind informacije o keš memoriji

Na slici 7 mogu se videti statistički podaci o backend delu aplikacije, a na slici 8 statistički podaci o frontend delu aplikacije ,gde je I broj instrukcija, I1 broj promašaja na prvom nivou keša instrukcija, LLi broj promašaja na poslednjem nivou keša instrukcija, D broj instrukcija, D1 promašaji, LLd promašaji, LL broj instrukcija, LL promašaji.

Na backend delu aplikacije broj promašaja keša instrukcija iznosi malo više od 2.5 miliona, što predstavlja 0.25% ukupnog referisanja keša instrukcija. Stopa promašaja na poslednjem nivou keša instrukcija iznosi 0.01%. Stopa promašaja na prvom i poslednjem nivou keša podataka je niska, posebno za čitanje podataka. Stopa promašaja keša poslednjeg nivoa je takođe niska, posebno za čitanje podataka. Rezultati profilisanja keš memorije na backend delu aplikacije sugerišu na niske stope promašaja keš memorije primećenih na različitim nivoima keš memorije, što ukazuje na njeno efikasno korišćenje, što doprinosi dobrim ukupnim performansama backend dela aplikacije, u smislu efikasnosti pristupa memoriji.

```
==8909== I   refs:      1,063,279,308
==8909== I1  misses:        2,691,957
==8909== LLi misses:          153,738
==8909== I1  miss rate:          0.25%
==8909== LLi miss rate:          0.01%
==8909==
==8909== D   refs:        474,600,873  (335,748,550 rd   + 138,852,323 wr)
==8909== D1  misses:          721,823  (    461,926 rd   +     259,897 wr)
==8909== LLd misses:          191,054  (     99,578 rd   +      91,476 wr)
==8909== D1  miss rate:           0.2% (        0.1%     +         0.2%  )
==8909== LLd miss rate:           0.0% (        0.0%     +         0.1%  )
==8909==
==8909== LL refs:           3,413,780  (  3,153,883 rd   +     259,897 wr)
==8909== LL misses:           344,792  (    253,316 rd   +      91,476 wr)
==8909== LL miss rate:            0.0% (        0.0%     +         0.1%  )
```

Slika 7: Statistički podaci o backend delu aplikacije

Na frontend delu aplikacije stopa promašaja na prvom nivou keša instrukcija iznosi 0.44%, a poslednjeg nivoa 0.23%. Stopa promašaja na prvom nivou keša podataka je visoka, posebno za čitanje podataka, dok je na poslednjem nivou keša podataka taj procenat manji. Stopa promašaja keša poslednjeg nivoa je niska, posebno za čitanje podataka. Rezultati profilisanja keš memorije na frontend delu aplikacije ukazuju na to da ima prostora za optimizaciju performansi keš memorije, kako bi se poboljšala ukupna efikasnost i performanse.

```
==8936== I   refs:      7,586,329,017
==8936== I1  misses:       33,645,263
==8936== LLi misses:       17,334,252
==8936== I1  miss rate:          0.44%
==8936== LLi miss rate:          0.23%
==8936==
==8936== D   refs:      2,790,063,032  (1,984,996,736 rd   + 805,066,296 wr)
==8936== D1  misses:       55,235,246  (   42,110,468 rd   +  13,124,778 wr)
==8936== LLd misses:       30,452,196  (   20,663,514 rd   +   9,788,682 wr)
==8936== D1  miss rate:           2.0% (          2.1%     +         1.6%  )
==8936== LLd miss rate:           1.1% (          1.0%     +         1.2%  )
==8936==
==8936== LL refs:          88,880,509  (   75,755,731 rd   +  13,124,778 wr)
==8936== LL misses:        47,786,448  (   37,997,766 rd   +   9,788,682 wr)
==8936== LL miss rate:            0.5% (          0.4%     +         1.2%  )
```

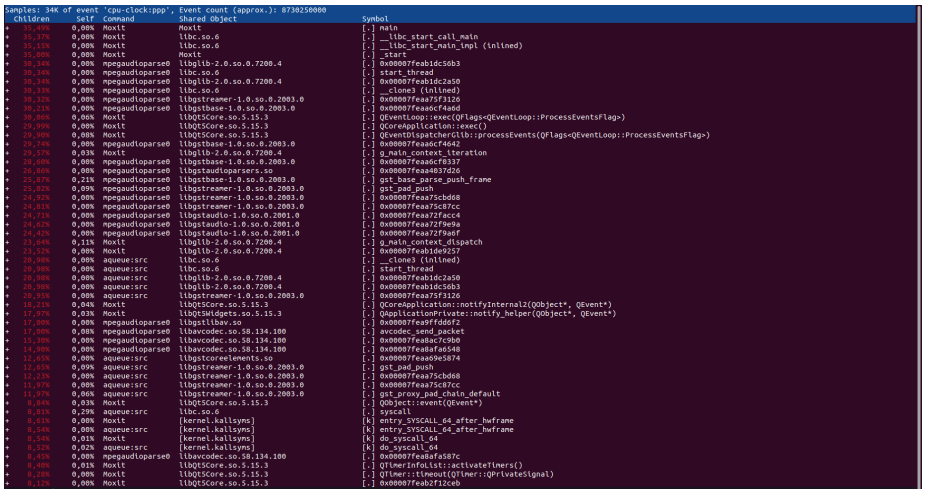
Slika 8: Statistički podaci o frontend delu aplikacije

3.2 Perf

Perf je alat za analizu performansi na Linux sistemima. Takođe može da prikuplja statističke uzorke izvršavanja programa u redovnim intervalima, omogućujući analizu ponašanja programa tokom vremena. Uzorkovanje može pomoći da se identifikuju uska grla u performansama i oblasti za optimizaciju.

Za lepši prikaz izlaza ovog profajlera korišćen je alat *FlameGraph*, koji rezultate prikazuje u vidu vatrenih grafika (eng. *flame graphs*). Na x osi je prikazana populacija uzoraka, a na y osi dubina steka. Slika vatrenog grafika može se pronaći u **perf/flamegraph.svg**.

Perf je na ovom projektu pokrenut na frontend delu aplikacije korišćenjem skripta **perf.sh**, a za dobijanje vatrenog grafika korišćena je skripta **flame_graph.sh**. Na slici 9 može se videti graf poziva koji prikazuje način na koji funkcije pozivaju jedna drugu tokom izvršavanja. Prilikom profilisanja frontend dela aplikacije prikupljeno je 34.000 uzoraka. Aproksimacija ukupnog broja događaja koji su se desili tokom profilisanja je više od 8.5 milijardi. Kolona *Children* predstavlja vreme izvršavanja funkcije i svih funkcija koje ona poziva, izraženo u procentima u odnosu na ukupno vreme izvršavanja programa. Kolona *Self* prestavlja vreme izvršavanja funkcije, izraženo u procentima u odnosu na ukupno vreme izvršavanja programa. Analizirajući graf poziva može se zaključiti da je vreme izvršavanja funkcije main i svih funkcija koje ona poziva najveće, što je i očekivano.



Slika 9: Graf poziva

4 Zaključak

Analizom ovog projekta može se zaključiti je da je on zaista kvalitetan. Uočeno je par propusta koji se tiču keš memorije na frontend delu aplikacije i curenja memorije na backend delu aplikacije, ali i pored toga projekat ima odlične performanse. Za svaki od primenjenih alata, dato je detaljno objašnjenje rezultata, kao i načina na koji se propusti mogu popraviti.