Analiza i Verifikacija Softvera

Online Shopping System

Dimitrije Marković 1022/24
Matematički fakultet, Univerzitet u Beogradu

25. septembar 2025.

Sadržaj

---

1 Uvod

Za potrebe kursa *Verifikacija Softvera*, analiziran je projekat **Online Shopping System**. Projekat je razvijen u programskom jeziku C++ kao konzolna aplikacija, sa klijent–server arhitekturom koja koristi `Winsock2` biblioteku za mrežnu komunikaciju.

Aplikacija implementira osnovne funkcionalnosti sistema za online kupovinu, uključujući registraciju i prijavu korisnika, pregled i kupovinu proizvoda, generisanje računa, kao i mogućnost izmene porudžbine. Dodatno, sistem pruža različite privilegije za korisnike, zaposlene i administratore (upravljanje nalozima, popusti, pregled inventara, rešavanje žalbi i automatsko procesuiranje porudžbina).

Cilj ove analize je da se proceni pouzdanost, efikasnost i kvalitet implementacije kroz upotrebu savremenih alata za statičku i dinamičku analizu softvera. U okviru rada korišćeni su sledeći alati:

- **Clang-Tidy** i **Cppcheck** — alati za statičku analizu koda i detekciju potencijalnih problema,
- **Doxygen** — za automatsko generisanje dokumentacije iz izvornog koda,
- **Dr. Memory** — za detekciju problema u radu sa memorijom tokom izvršavanja,
- **Very Sleepy Profiler** — za profilisanje performansi i identifikaciju najintenzivnijih delova koda.

Analiza i dobijeni rezultati omogućavaju uvid u potencijalne slabosti projekta i daju preporuke za dalje unapređenje koda, sa ciljem povećanja njegove stabilnosti, efikasnosti i dugoročne održivosti.

2 Statička analiza

2.1 Clang-Tidy

2.1.1 Uvod

Alat **Clang-Tidy** je korišćen za proveru koda u direktorijumima `Server` i `Client`. Pokretanje je vršeno pomoću batch skripte koja rekurzivno prolazi kroz sve `.cpp` fajlove i primenjuje sledeće grupe provera: `clang-diagnostic-*`, `clang-analyzer-*`, i `modernize-*`.

2.1.2 Sažetak rezultata

- **Greške (errors):** 2 tipa, sa više pojavljivanja.
- **Upozorenja (warnings):** oko 8000 ukupno.

2.1.3 Greške (errors) i rešenja

(E1) Nedefinisan `std::basic_ostringstream<char>` **(#include** `<sstream>`**)** **Opis:** Greška nastaje kada se koristi `std::ostringstream` bez uključivanja hedera `<sstream>`.

Fajlovi (reprezentativno):

- `Server/Headers/Admin.cpp`,
- `Server/Headers/Employee.cpp`,
- `Server/Headers/Person.cpp`,

- `Server/Headers/Server.cpp`,
- `Server/Headers/Customer.cpp`.

Rešenje: Dodati sledeći include u svaku prevodnu jedinicu gde se koristi `ostringstream`:

```
#include <sstream>
```

(E2) Konflikt Winsock zaglavlja (redefinicije tipova / expected identifier) Opis: Pojavljuju se višestruke greške (*"expected identifier"*, *"redefinition of 'sockaddr'/'fd_ set' ..."*) koje potiču iz Windows SDK hedera (`ws2def.h`, `winsock2.h`). Uzrok je uključivanje `<windows.h>` pre `<winsock2.h>`, što povlači zastareli `winsock.h` i izaziva konflikt.

Fajlovi (posredno): greške su prijavljene u SDK hederima, ali izvorni kod se nalazi u mrežnim modulima, npr. `Server/Headers/Thread.cpp`, `Server/Headers/Server.cpp`.

Rešenje: Osigurati ispravan redosled uključivanja u svim fajlovima koji koriste mrežu:

```
#define WIN32_LEAN_AND_MEAN // spreava povlaenje suvinih API-ja
#include <winsock2.h> // uvek pre <windows.h>
#include <ws2tcpip.h> // za InetPton/InetNtop
#include <windows.h> // tek posle winsock2.h
```

2.1.4 Upozorenja (warnings) i preporuke

(W1) Zastarele/nesigurne CRT i Winsock funkcije Opis: Korišćenje zastarelih ili nesigurnih funkcija.

Primeri:

- `getch()` → koristiti `_getch()` (`<conio.h>`),
- `strcpy()` → koristiti `strcpy_s()` ili preći na `std::string`,
- `ctime()`, `localtime()` → koristiti `ctime_s()`, `localtime_s()`,
- `inet_addr()`, `inet_ntoa()` → koristiti `InetPton()`, `InetNtop()`.

Fajlovi: `Client/Headers/Person.cpp`, `Client/Headers/Admin.cpp`, `Client/Headers/Customer.cpp`, `Server/Headers/Employee.cpp`, `Server/Headers/Complaint_Base.cpp`, `Server/Headers/Complaint_E.cpp`, `Server/Headers/Server.cpp`, `Server/serverProgram.cpp`, `Client/clientProgram.cpp`. .

(W2) Modernizacija C++ koda (modernize checks)

- `modernize-use-nullptr` (˜15 puta): zameniti `NULL` sa `nullptr`. **Fajlovi:** `Server/Headers/Server.h`, `Server/Headers/Thread.cpp`, `Server/Headers/Server.cpp`, `Client/Headers/Person.cpp`.
- `modernize-loop-convert` (˜8 puta): zameniti klasične `for` petlje sa range-based petljama `for`.

  Fajlovi: `Server/Headers/Person.cpp`, `Server/Headers/Admin.cpp`.

(W3) Sigurnosna upozorenja (`clang-analyzer-security.insecureAPI.strcpy`) **Opis:** Korišćenje `strcpy` je prijavljeno kao nesigurno. **Fajlovi:** `Server/Headers/Complaint_Base.cpp`, `Server/Headers/Complaint_E.cpp`.

Rešenje: koristiti `strcpy_s()` sa granicama bafera ili preći na `std::string`.

(W4) Neispravan poziv funkcije **Opis:** U `Server.cpp` je pozvana funkcija `WSAGetLastError` bez zagrada, što ispisuje pokazivač umesto vrednosti.

Rešenje:

```
std::cerr << "Error:␣" << WSAGetLastError() << std::endl;
```

(W5) Potencijalno neinicijalizovani objekti **Opis:** Detektovan je neinicijalizovani član u klasi `Client`.

Fajl: `Client/Headers/Client.h`.

Rešenje: inicijalizovati sve članove u konstruktoru, npr:

```
Client(const std::string& ipAddr = "127.0.0.1", int port = 7777)
  : m_ipAddress(ipAddr), m_port(port), client(INVALID_SOCKET) {}
```

2.1.5 Zaključak analize clang-tidy-a

Greške: najkritičniji problemi su nedostajući `<sstream>` i konflikt Winsock zaglavlja.

Upozorenja: odnose se na zastarele i nesigurne funkcije, potrebu za modernizacijom koda, sigurnosne rizike sa `strcpy`, neispravne pozive funkcija i neinicijalizovane članove. Implementacijom predloženih rešenja smanjiće se rizik od sigurnosnih propusta i poboljšaće se čitljivost i stabilnost koda.

2.2 Cppcheck

2.2.1 Uvod

Alat **Cppcheck** je korišćen za statičku analizu koda sa ciljem da se otkriju potencijalni problemi koji nisu uvek detektovani kompajlerom. Analiza je obuhvatila i `Server` i `Client` deo projekta.

2.2.2 Glavne kategorije detektovanih problema

- **Nedostajući sistemski hederi (missingIncludeSystem)**
  Prijavljeno u velikom broju fajlova, npr. `Server/Headers/Person.h`, `Server/Headers/Server.h`, `Client/Headers/Person.h`, `Client/Headers/Client.h`.
  Ovi upozorenja ne predstavljaju realan problem jer Cppcheck ne zahteva standardne biblioteke, ali ukazuju na zavisnost projekta od platforme (Windows-specifični hederi poput `conio.h`, `windows.h`, `ws2tcpip.h`).

- **Neinicijalizovane promenljive (uninitMemberVar)**
  `Thread::data` u `Server/Headers/Thread.h`, kao i `Person::sex`, `Person::age`, i `Person::DOB` u `Client/Headers/Person.h`.
  Rešenje: dodati inicijalizaciju u konstruktorima:

  ```
  Person::Person() : sex('U'), age(0), DOB("") {}
  ```

- **Klase bez konstruktora (noConstructor)**
  Klase `Goods`, `Cash`, `Complaint_Base`, `Complaint_E` nemaju deklarisan konstruktor iako poseduju privatne članove.
  Rešenje: eksplicitno deklarisati konstruktore i inicijalizovati sve članove.

- **Pogrešan redosled u initializer listi (initializerList)**
  Detektovano u `Server::m_port` (`Server/Headers/Server.h`). Članovi klase se uvek inicijalizuju redosledom deklaracije.
  Rešenje: prilagoditi redosled initializer liste redosledu deklaracije članova.

- **Funkcije koje mogu biti `const` (functionConst)**
  Na primer: `Admin::view`, `Admin::accounts`, `Cash::get_final_cash`, `Complaint_Base::view`, `Complaint_Base::see`, `Person::transfer_to_file`.
  Rešenje: dodati ključnu reč `const` kada funkcija ne menja unutrašnje stanje objekta.

- **Funkcije koje mogu biti `static` (functionStatic)**
  Detektovano u `Person::initialize_goods`, `Person::initialize_cash`, `Server::updateActivity`, `Server::getActivity`, `Server::closeSocket`.
  Rešenje: ako funkcija ne koristi stanje objekta, može biti deklarisana kao `static` radi performansi.

- **Konstruktori koji nisu `explicit` (noExplicitConstructor)**
  Klase `Server`, `Employee`, `Admin`, `Customer`, `Client`.
  Rizik: implicitne konverzije.
  Rešenje: koristiti `explicit` za jednoargumentne konstruktore.

- **Opseg promenljivih (variableScope)**
  Promenljive poput `ans`, `previous`, `Balance` imaju širi opseg nego što je potrebno (`Admin.cpp`, `Complaint_Base.cpp`, `Person.cpp`).
  Rešenje: deklarisati promenljive u bloku u kom se koriste.

- **Problemi sa prenosom argumenata (constParameterReference)**
  Na primer: parametri funkcija `Person::update_cash`, `Thread::Rec`, `Thread::Client` mogu biti prosleđeni kao `const &`.
  Rešenje: koristiti `const T&` za velike objekte da bi se izbeglo nepotrebno kopiranje.

- **Nepotrebne ili sumnjive operacije (constStatement)**
  Detektovano u `Admin.cpp`, `Person.cpp` (izrazi tipa `+` ili `-` bez efekta).
  Rešenje: proveriti da li je to bug ili nepotreban kod.

- **Problemi sa prenosom argumenata funkcija (funcArgNamesDifferent)**
  Na primer: `search` u `Admin.cpp` koristi različita imena parametara u deklaraciji i definiciji (`str` vs `key`).
  Rešenje: Staviti ista imena parametara u .c i .h fajlu.

- **Portabilnost: korišćenje `fflush(stdin)`**
  Detektovano u `Admin.cpp` (više puta). Ovo ponašanje nije definisano na svim sistemima.
  Rešenje: koristiti druge metode za čišćenje bafera ulaza (`cin.ignore()` u C++).

- **Stilski problemi**
  - **C-style kastovi** (`(type)ptr`) u `Server.cpp`. Preporuka je korišćenje C++ kastova (`static_cast`, `reinterpret_cast`).
  - **Unused variables** (`index` u `Admin.cpp`). Ukloniti ako nisu potrebne.

2.2.3 Zaključak analize Cppcheck-a

Analiza je otkrila značajan broj stilskih i sigurnosnih problema. Najkritičniji su neinicijalizovani članovi klasa (`Person`, `Thread`), upotreba nesigurnih funkcija (`fflush(stdin)`), kao i implicitni konstruktori bez `explicit` modifikatora. Preporučuje se uvođenje konstruktora sa inicijalizacijom, dodavanje `explicit`, kao i korišćenje modernih C++ tehnika (`const`, `auto`, range-based petlje) kako bi se unapredio kvalitet i stabilnost koda.

3 Doxygen

3.1 Uvod

Doxygen je alat koji omogućava automatsko generisanje tehničke dokumentacije iz izvornog koda na osnovu komentara napisanih u skladu sa njegovom sintaksom. Korišćen je u ovom projektu kako bi se dobio jasan pregled strukture sistema, odnosa između klasa i funkcija, kao i tokova poziva unutar aplikacije.

Generisana dokumentacija olakšava razumevanje postojećeg koda i može poslužiti kao osnova za dalje održavanje i proširenje sistema. Posebno je korisna u projektima većeg obima gde je potrebno brzo sagledati strukturu i interakcije komponenti.

3.2 Generisanje dokumentacije

Nakon podešavanja konfiguracionog fajla `Doxyfile`, dokumentacija se generiše jednostavnom komandom:

```
doxygen Doxyfile
```

Pre pokretanja je neophodno u konfiguracionom fajlu definisati osnovne parametre kao što su naziv projekta, uključivanje poddirektorijuma, izbor formata izlaza (HTML, LaTeX), i opcije za generisanje dijagrama zavisnosti. Početna stranica dokumentacije nalazi se u fajlu `html/index.html`.

3.3 Struktura generisane dokumentacije

Generisana HTML dokumentacija sadrži sledeće glavne celine:

- **Main Page** – osnovne informacije o projektu.
- **Classes** – detaljan pregled svih klasa i struktura, uključujući:
  - **Class List** – lista svih klasa i struktura u projektu.
  - **Class Hierarchy** – hijerarhijski prikaz odnosa baznih i izvedenih klasa.
  - **Class Members** – objedinjeni pregled svih članova (funkcije, promenljive).
- **Files** – lista svih izvornog koda i heder fajlova, sa povezanim klasama i funkcijama.
- **Search** – polje za pretragu.

3.4 Zapažanja i problemi

Tokom analize uočeno je da kod nije dosledno komentarisán, a na mestima gde komentari postoje ne prate Doxygen standard. To otežava generisanje kvalitetne dokumentacije.

Primer ispravnog Doxygen komentara:

```
/**
 * @brief Izracunava ukupan iznos porudzbine.
 * @param products Lista proizvoda u korpi.
 * @return Ukupan iznos porudbine.
 */
double calculateTotal(const std::vector<Product>& products);
```

Preporuka je da se svi javni interfejsi (klase, metode, promenljive) dokumentuju ovim formatom. Pored toga, mogu se koristiti i dodatne oznake:

- `@note` – napomena o specifičnostima funkcije,

- `@warning` – upozorenja o potencijalnim problemima,
- `@todo` – stavke koje je potrebno naknadno implementirati ili doraditi.

3.5 Napredne funkcionalnosti

Aktiviranjem opcija `CALL_GRAPH` i `CALLER_GRAPH`, generisani su dijagrami koji pružaju vizuelni prikaz toka izvršavanja:

- **Call Graph** – prikazuje koje funkcije određena metoda poziva.
- **Caller Graph** – prikazuje iz kojih funkcija se određena metoda poziva.

Ovi grafovi su se pokazali kao veoma korisni za razumevanje toka rada i međusobne povezanosti različitih delova koda, posebno u složenijim sistemima.

4 Very Sleepy Profiler

4.1 Uvod

Very Sleepy je sampling profiler za Windows. U ovoj analizi profilisane su odvojeno `server.exe` i `client.exe`. Podaci su eksportovani u `.callgrind` i `.csv` formate i korišćeni za identifikaciju *hotspot* funkcija i kritičnih putanja poziva.

- **Profilisanje:** sampling (*stack snapshots*); trajanje i interval uzorkovanja prema podrazumevanim postavkama Very Sleepy.
- **Artefakti:** `server.callgrind`, `server.csv`, `client.callgrind`, `client.csv`.
- **Napomena:** CRT start funkcije (`__tmainCRTStartup`, `mainCRTStartup`) su očekivano visoko u listi zbog ulazne tačke procesa i ne predstavljaju cilj optimizacije.

4.2 Rezultati (Server)

Tabela 1: Server — projektne funkcije koje najviše vode do mrežnog hotspota

Caller	Inclusive %	Poziva
`worker`	50.15	104
`Server::run`	50.15	55
`Thread::Rec`	50.15	11
`Person::login`	50.15	20
`Employee::complain`	50.15	10
`Employee::home`	27.01	6
`Admin::home`	23.13	6
`Customer::start`	20.03	12

Tabela 2: Server — sistemske/CRT putanje do mrežnog hotspota

Caller	Inclusive %	Poziva
`select (WS2_32)`	50.15	41
`recv (WS2_32)`	50.15	6
`NSPStartup (mswsock)`	50.15	1
`Tcpip4_WSHAddressToString (mswsock)`	49.85	3
`mainCRTStartup`	50.15	81
`__tmainCRTStartup`	50.15	81
`main`	50.15	81

Zapažanja (Server).

- **Mrežni sloj dominira:** `Server::run`/`Thread::Rec` troše najviše vremena ⇒ trošak je fokusiran na petlju koja prihvata/obradjuje i čita sa soketa.
- **Sistemske funkcije:** `NSPStartup`/`Tcpip4_WSHAddressToString` su spoljne (OS/network) rutine, nisu primarni kandidati za refaktorisanje.
- **I/O i fajlovi:** pojavljuju se `std::basic_file*::open` (zanemarljiv udeo).

4.3 Rezultati (Client)

Tabela 3: Client — projektne funkcije sa najvećim udelom

Funkcija	Inclusive %
`Person::login`	90.85
`Admin::home`	41.52
`Employee::complain`	29.55
`Employee::home`	28.32
`Customer::start`	19.32
`Customer::home`	16.26
`Person::buy`	11.56
`Customer::complain`	7.91
`Admin::search`	6.95
`Admin::balance`	4.60
`Admin::accounts`	0.45

Tabela 4: Client — standardne/sistemske rutine koje dominiraju (Inclusive %)

Funkcija	Inclusive %
`fgetc` (msvcrt)	90.72
`__gnu_cxx::stdio_sync_filebuf<...>::underflow`	85.82
`std::istream::operator»`	59.57
`std::istream::sentry::sentry`	58.84
`std::getline<char, std::char_traits<char>, std::allocator<char> >`	26.31
`__gnu_cxx::stdio_sync_filebuf<...>::uflow`	6.40
`std::istream::get`	4.89
`fprintf` (msvcrt)	0.55

Zapažanja (Client).

- **Ulaz sa konzole:** veliki udeo vremena u

  `stdio_sync_filebuf::underflow` i `std::istream::sentry::sentry`

  ukazuje da su `std::cin`/konzolni ulazi glavni *hotspot*.

- **Tokovi aplikacije:** `Person::login`/`Customer::start`/`Person::buy` čine glavne korisničke scenarije.

Napomena

Prikazani procenti i vremena su rezultat *sampling* profilisanja, te predstavljaju procene zasnovane na uzorkovanim stanjima steka u datom scenariju i okruženju (trajanje, interval uzorkovanja, build konfiguracija). Kao takvi, pouzdano ističu *hotspot* funkcije za konkretno opterećenje, ali nisu apsolutne metrike važeće za sve slučajeve.

Statičke optimizacije iz odeljka Clang-Tidy (npr. zamena `strcpy`, `_getch`, `inet_*` → `InetPton/Ntop`) i modernizacije (`nullptr`, range-based `for`) primarno unapređuju ispravnost, bezbednost i održivost koda; njihov efekat na performanse je posredan i slučajan. Nasuprot tome, **profilisanje** ukazuje gde se troši najveći deo *uzorkovanog* CPU vremena u realnom radu, pa stoga služi za ciljanje optimizacija nakon što je kod stabilizovan statičkim ispravkama.

5 Dr. Memory

5.1 Uvod

Alat **Dr. Memory** je korišćen za dinamičku analizu izvršavanja servera i klijenta. Alat detektuje curenja memorije, neinicijalizovane ili nevažeće pristupe memoriji, *use-after-free*, dvostruko oslobađanje i oštećenja heap-a.

5.2 Dr. Memory — analiza servera

Rezultati. U fajlovima sa rezultatima zabeležen je velik broj prijava . Najčešće kategorije su:

- **UNINITIALIZED READ** : čitanja neinicijalizovanih registara/memorije (npr. `eax`, `r9`, `eflags`) u `WSAStartup`/`accept`/`closesocket`, pozivane iz

  `Server::start`/`Server::Listen`/`main`.

- **UNADDRESSABLE ACCESS**: pristup *beyond top of stack* u toku `Server::start`.

- **Potential errors (blocklist)**: deo UNINIT čitanja u `KERNELBASE.dll`/`WS2_32.dll` alat klasifikuje kao verovatne lažno pozitivne.

Diskusija grešaka. Dominantne prijave su neinicijalizovana čitanja u WinSock toku, tipično:

- `replace_memset` → `WSAStartup` → `Server::start` → `main` — UNINIT pri inicijalizaciji WinSock-a.

- `WS2_32.dll!closesocket` → `Server::Listen` → `Server::start` → `main` — UNINIT na registru `r9` pri zatvaranju konekcije.

- *Beyond top of stack* — upućuje na pogrešnu upotrebu lokalnih bafera/pokazivača ili prelazak opsega.

Preporuke za ispravku.

1. **Inicijalizacija i provere:** pre svakog WinSock poziva (`WSAStartup`/`accept`/`closesocket`) inicijalizovati strukture (`sockaddr_in`, `WSADATA`) i proveriti povratne kodove.
2. **RAII za sokete i memoriju:** uvesti omotače koji automatski zatvaraju `SOCKET` u destruktoru; posle `closesocket` postaviti deskriptor na `INVALID_SOCKET`.
3. **Životni vek objekata:** ne vraćati/prosleđivati pokazivače na lokalne bafer-e; za veće strukture koristiti dinamičku alokaciju ili statički trajni objekat.

Zaključak. Nisu uočena tipična curenja ili dvostruka oslobađanja, ali su detektovana brojna UNINIT čitanja i jedno oštećenje steka. Uz doslednu inicijalizaciju, RAII upravljanje soketima i reviziju upotrebe lokalnih bafera, stabilnost serverskog dela se značajno unapređuje.

5.3 Dr. Memory — analiza klijenta

Rezultati. Analiza više pokretanja (*clientProgram.exe*) pokazuje ponovljiv obrazac:

- **UNINITIALIZED READ** tokom WinSock inicijalizacije i DNS/servisnih upita:

  `WSALookupServiceNextW`/`WSAStartup` (registri `eax`, `eflags`; čitanja iz memorijskih slotova oko `0x7ef930`).
- **UNINITIALIZED READ** pri gašenju/rezoluciji: `closesocket` (reg. `r9`) i `getnameinfo`/`WSAUnhookBlockingHook` (reg. `rcx`).
- **UNADDRESSABLE ACCESS** u ranoj CRT/relocation fazi (*beyond top of stack*), pre korisničke logike.
- **Potential errors (blocklist)**: deo UNINIT prijava iz `KERNELBASE.dll`/`WS2_32.dll` klasifikovan kao OS/CRT šum.

Diskusija grešaka. Ključne tačke:

- `Client::start` → `main` — UNINIT u `WSAStartup`/`WSALookupServiceNextW` tipično nastaje kada strukture/parametri nisu nulto inicijalizovani ili se oslanjaju na nedefinisane vrednosti.
- `Client::Connect` — UNINIT na `r9` pri `closesocket` sugeriše dvostruko zatvaranje ili upotrebu nevalidnog `SOCKET`-a; UNINIT u `getnameinfo` ukazuje na ulazne parametre/dužine.
- *Beyond top of stack* u CRT startupu najverovatnije nije iz korisničkog koda, ali opravdava proveru build okruženja/flagova i upotrebe velikih lokalnih bafera.

Preporuke za ispravku.

1. **Deterministička inicijalizacija:** nulto inicijalizovati `WSADATA`, `sockaddr_in`/`addrinfo` i sve pomoćne strukture; validirati `len` vrednosti pre poziva API-ja.
2. **Bezbedno rukovanje soketima:** RAII omotač za `SOCKET`; pre `closesocket` proveriti `socket != INVALID_SOCKET`, a potom setovati `INVALID_SOCKET`.
3. **Ulazno/izlazni baferi:** koristiti `std::vector<char>` ili `std::array` i osigurati potpuni inicijalizacioni opseg pre `send`/`recv`/`getnameinfo`.

Zaključak. Klijent deli obrazac sa serverom: dominiraju UNINIT čitanja u WinSock stazama, uz nekoliko relevantnih tačaka u korisničkom kodu (zatvaranje soketa, validacija parametara i bafera). Primena preporuka (inicijalizacija, RAII, proverene dužine/parametri, bezbedni baferi) umanjuje rizik od oštećenja memorije i povećava pouzdanost klijentskog dela.

6 Zaključak

Analiza projekta **Online Shopping System** kombinovanjem statičkih alata (*Clang-Tidy*, *Cppcheck*), generisane dokumentacije (*Doxygen*), dinamičke analize memorije (*Dr. Memory*) i profajlisanja performansi (*Very Sleepy*) obezbedila je sveobuhvatan uvid u kvalitet i pouzdanost implementacije.

Pozitivni aspekti. Arhitektura klijent–server je jasno razdvojena i funkcionalno koherentna. Doxygen dokumentacija (hijerarhija tipova, call grafovi) olakšava razumevanje uloga ključnih komponenti, dok rezultati profilisanja potvrđuju da se najveći deo vremena troši u očekivanim tačkama (mrežni I/O na serveru, konzolni I/O na klijentu), bez neočekivanih uskih grla.

Uočeni problemi. Statička analiza ukazuje na upotrebu zastarelih/nesigurnih API-ja i na potrebu modernizacije C++ stila (npr. `nullptr`, `explicit`, `const`-ispravnost, range-based `for`), kao i na pojedine neinicijalizovane članove i nedosledan redosled u *initializer* listama. Dinamička analiza detektuje *UNINITIALIZED READ* događaje i pristupe izvan važećeg opsega stoga u stazama vezanim za `Winsock` i lokalne bafer; deo tih nalaza potiče iz sistemskih biblioteka i može biti blok-listiran, ali deo ukazuje na realne rizike u tretmanu `SOCKET` deskriptora i validaciji bafera pri `send`/`recv`.

Preporuke.

- Ispravan redosled uključivanja zaglavlja (`winsock2.h` pre `windows.h`, `WIN32_LEAN_AND_MEAN`) i zamena zastarelih funkcija sigurnijim alternativama (`strcpy` → `strcpy_s`/`std::string`, `getch` → `_getch`, `inet_*` → `InetPton/Ntop`).
- RAII pristup za sokete i resurse (jednoznačno vlasništvo; posle `closesocket` postaviti `INVALID_SOCKET`).
- Inicijalizacija i validacija svih struktura/bafera pre mrežnih poziva; izbegavanje prosleđivanja pokazivača na objekte sa ograničenim životnim vekom.
- Postupna modernizacija koda radi čitljivosti i održivosti: `explicit` konstruktori, `const`-korektnost, bezbedni kastovi, sužavanje opsega promenljivih.

Primena preporučenih izmena podiže pouzdanost sistema, olakšava buduće održavanje i usmerava dalji razvoj na najuticajnije delove koda.