Politechnika Warszawska, WEiTI

pRoman - process manager

Projekt TIN. Dokumentacja końcowa

Damian Bułak
Michał Sypetkowski
Marcin Waszak
Ahata Valiukevich

1 czerwca 2017

1 Projekt wstępny

1.1 Treść zadania

W sieci jest zbiór zarządzanych węzłów, serwer zarządzający i stacja konsoli administratora. W węzłach pracują agenty zarządzające. Agent zarządzający może: załadować kod nowego procesu, usunąć kod procesu, uruchomić/zatrzymać/wznowić/zabić dany proces zgodnie z harmonogramem, wznowić proces nie raportujący swej żywotności, podać dane statystyczne serwerowi. System umożliwia administratorowi zarządzanie rozproszonymi procesami. System komunikacji powinien móc pracować w przestrzeni adresów *IPv4* i *IPv6*. Ponadto należy zaprojektować moduł do *Wireshark* umożliwiający wyświetlanie i analizę zdefiniowanych komunikatów.

1.2 Przyjęte założenia funkcjonalne i niefunkcjonalne

Założenia funkcjonalne

Implementowany system ma umożliwiać:

- wysyłanie obrazu procesu
- usuwanie obrazu procesu

- wznawianie procesu nie raportującego swej żywotności
- podawanie danych statystycznych serwerowi
- uruchamianie danego procesu zgodnie z harmonogramem
- zatrzymywanie danego procesu zgodnie z harmonogramem
- wznawianie danego procesu zgodnie z harmonogramem
- zabijanie danego procesu zgodnie z harmonogramem

Założenia niefunkcjonalne

- niezawodność komunikacji dzięki użyciu protokołu TCP
- funkcja watchdoga chroniącego przed zawieszaniu się agenta
- intuicyjna obsługa dzięki zastosowaniu poleceń z poziomu konsoli

1.3 Podstawowe przypadki użycia

Administrator - użytkownik zalogowany do konsoli administratorskiej

Tablica 1: Podstawowe przypadki użycia

NAZWA	AKTORZY	OPIS
Wysłanie obrazu procesu	Administrator	1. Administrator wybiera proces do załadowania. 2. Proces zostaje załadowany do systemu.
Przeglądanie listy procesów	Administrator	1. Administratorowi prezentowana jest lista załadowanych procesów. 2. Obok każdego procesu administrator może zobaczyć stan procesu (uruchomiony bądź nie), jak i podstawowe informacje o procesie.

Kontynuacja na następnej stronie

Tablica 1 – *Kontynuacja z poprzedniej strony*

NAZWA	AKTORZY	OPIS
Usuwanie obrazu procesu	Administrator	1. [Przeglądanie listy procesów] 2. Administrator usuwa wybrany z listy proces poleceniem *'Usuń'*.
Uruchomienie procesu	Administrator	1. [Przeglądanie listy procesów] 2. Administrator wybiera z listy proces i uruchamia go poleceniem *'Uruchom'*. 3. Następuje przetwarzanie procesu (opis w architekturze systemu). Administrator może zobaczyć postęp przetwarzania procesu w postaci procent zakończenia. 4. Po zakończeniu przetwarzania administrator widzi rezultat przetwarzania.
Zatrzymanie uruchomionego procesu	Administrator	1. [Przeglądanie listy procesów] 2. Administrator wybiera z listy uruchomiony proces. 3. Administrator zatrzymuje proces poleceniem *'Zatrzymaj'*.
Planowanie czasu uruchomienia procesu	Administrator	1. [Przeglądanie listy procesów] 2. Administrator edytuje czas wykonania procesu po wpisaniu polecenia *'Edytuj'*.

Kontynuacja na następnej stronie

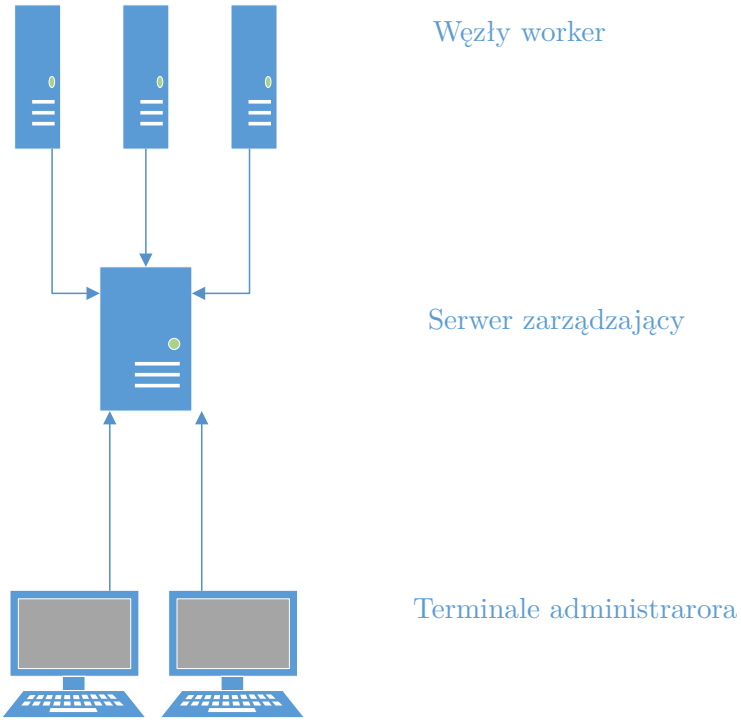
Tablica 1 – *Kontynuacja z poprzedniej strony*

NAZWA	AKTORZY	OPIS
Podgląd harmonogramu procesów	Administrator	1. Administrator przegląda tabelaryczny harmonogram załadowanych procesów.
Podgląd danych statystycznych	Administrator	1. Administrator przegląda dane statystyczne z informacjami o załadowanych, uruchomionych, zatrzymanych procesach oraz o obciążeniu poszczególnych węzłów.

1.4 Wybrane środowisko sprzętowo-programowe

1. System operacyjny: ***GNU/Linux***.
2. Język: ***C++***.
3. Biblioteki: STL, boost/filesystem, boost/process, BSD Sockets API, GNU Readlines.
4. Testowanie: python3.
5. Debugowanie: GDB, Wireshark.
6. Budowanie: cmake

1.5 Architektura rozwiązania



Rysunek 1: Ilustracja systemu

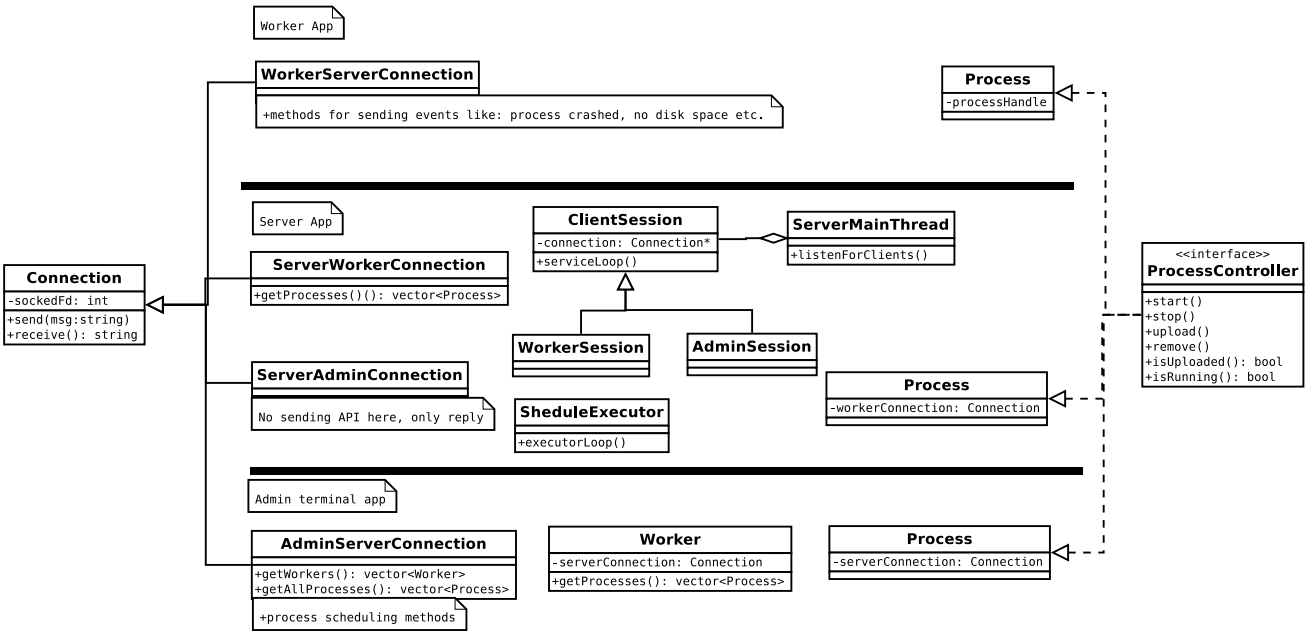
Węzeł worker - maszyna, która wykonuje procesy. Bezpośrednia kontrola procesów jest sprawowana przez procesy sieciowe klasy worker.
Proces klasy worker- specjalny proces sieciowy na maszynach węzłów worker. Jego zadaniem jest zarządzanie procesami na tych maszynach według rozkazów serwera zarządzającego.
Watchdog - specjalny proces obok procesu workera. Stanowi zabezpieczenie przed skutkami zawieszenia procesu workera.
Serwer zarządzający - maszyna, która wysyła rozkazy do agentów maszyn węzłowych. Może przeprowadzać zaplanowane wcześniej czynności. Serwer posiada proces sieciowy klasy server.
Proces klasy server - oczekuje na połączenia od węzłów worker i terminali administratora.
Terminal administratora - maszyna, dzięki której administrator ma możliwość zarządzać procesami na maszynach węzłowych. Każdy taki terminal posiada proces sieciowy klasy administrator.

Planowane aplikacje sieciowe do wykonania:

- aplikacja klasy worker

- aplikacja klasy server
- aplikacja klasy administrator

1.6 API modułów stanowiących bloki funkcjonalne



Rysunek 2: Wstępny diagram klas

1.7 Sposób testowania

Planujemy użyć testów jednostkowych w poszczególnych aplikacjach naszego projektu, żeby sprawdzić poprawność działania osobnych części całego systemu. Poza testami jednostkowymi napisane będą również testy integracyjne. Przewidujemy rownież testowanie ręczne. Scenariusze testów akceptacyjnych są przedstawione poniżej.

1.8 Scenariusze testów akceptacyjnych

Testy akceptacyjne będą polegały na wykonywaniu różnych scenariuszy użycia.

Tablica 2: Przykładowe scenariusze testów akceptacyjnych

CEL TESTU	OPIS PRZEBIEGU	OCZEKIWANY WYNIK
Usuwanie obrazu procesu	Administrator przegląda listę procesów. Wpisuje polecenie do usunięcia procesu oraz jego id.	Jeśli podane id istnieje, obraz procesu zostanie usunięty, w przeciwnym przypadku pojawi się komunikat o braku danego procesu.
Zatrzymanie uruchomionego procesu	Administrator wpisuje polecenie zatrzymania procesu.	Administrator otrzymuje informację o zatrzymaniu procesu.Proces jest widoczny na liście procesów jako załadowany (nie uruchomiony). Jeśli proces nie został jeszcze uruchomiony administrator otrzymuje informację o błędzie.
Załadowanie obrazu procesu	Administrator wpisuje polecenie załadowania nowego obrazu procesu.	Informacja o stanie załadowania procesu - sukces lub nieudane załadowanie, wraz ze wskazaniem przyczyn niepowodzenia w wykonaniu polecenia.
Podgląd harmonogramu procesów	Administrator wpisuje polecenie przeglądu procesów.	Wyświetlona lista procesów wraz z ich planowanymi czasami uruchomienia.
Podgląd danych statystycznych	Administrator wpisuje polecenie służące do podglądu danych statystycznych serwera.	Wyświetlone zostaną informacje o obciążeniu węzłów, liczbie uruchomionych procesów, liczbie załadowanych procesów etc.

1.9 Podział prac w zespole

Damian: projekt architektury
Michał: projekt architektury
Agata: projekt architektury, dokumentacja, testowanie, moduł Wireshark
Marcin: team-leader, projekt architektury

Powyższy podział jest zgrubny. Bowiem szczegółowe przydzielanie prac zorganizowane zostało za pomocą systemu tasków na platformie GitHub.

1.10 Harmonogram prac

- **19.04** - oddanie dokumentacji wstępnej.

- **30.04** - szkielet programów: serwera, workera oraz administratora; rozpoczęte prace nad skryptami testującymi.
- **8.05** - konsultacje #1: projekt z w pełni działającą komunikacją między administratorem a serwerem (nawiązywanie połączenia, działający mechanizm przesyłania pakietów i poleceń).
- **22.05** - konsultacje #2: działający i prawie skończony projekt, z ewentualnymi błędami niewpływającymi istotnie na działanie programu, rozpoczęcie pracy nad końcową dokumentacją poprojektową.
- **31.05** - oddanie projektu.

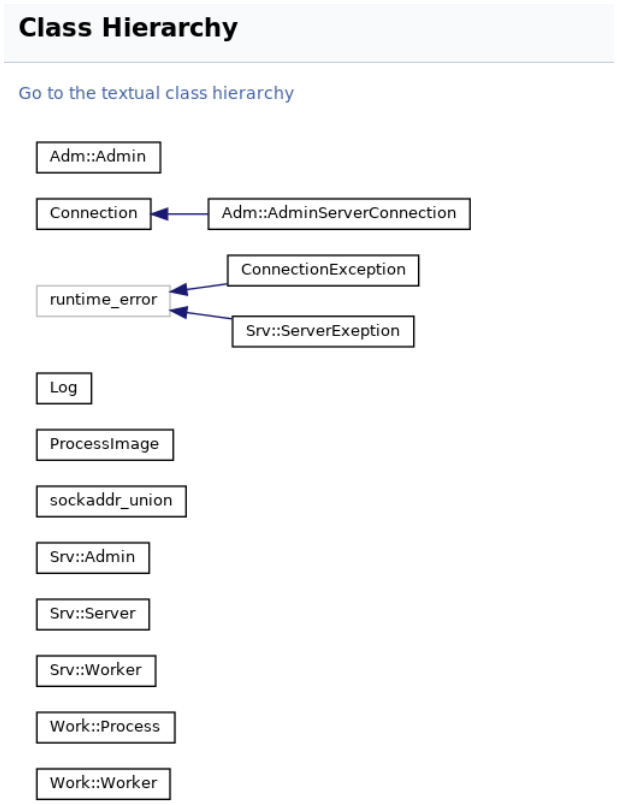
1.11 Adres projektu na serwerze kontroli wersji

https://github.com/marcin-waszak/DistributedProcesses

W zakadce *Issues* można przeglądać informacje o przydzielanych przez Marcina zadaniach:

- dla kogo zostało przydzielone
- kiedy zostało utworzone
- na jakim etapie realizacji obecnie jest

2 Ilustracja i opis struktury implementacyjnej systemu



Rysunek 3: Diagram klas

3 Definicje komunikatów

Struktura i sposób ich kodowania

Każdy pakiet przesyłany pomiędzy modułami systemu składa się z:

- długości komendy - 4 bajty
- metody
- długości danych - 4 bajty
- dane

Przy czym dwa ostanie elementy mogą występować wielokrotnie. Wewnętrznie system za każdym razem wysyła długość danych do odebrania i daną. Przy czym komenda jest w tym rozumieniu specjalnym rodzajem danej. Przykładowo komunikat **UPLOAD_IMAGE abcd** będzie skutkował wysłaniem pakietu (pomijając opakowanie w warstwę transportową itd.): **12UPLOAD_IMAGE4abcd**.

Określenie kto jakie komunikaty odbiera i wysyła

Wyróżniono następujące komunikaty:

- Wysyłane ze stacji administratora do serwera:
  - **ADMIN** - nawiązanie połączenia z serwerem, serwer od momentu otrzymania takiego komunikatu identyfikuje administratora
  - **GET_WORKERS** - serwer zwraca administorowi listę dostępnych stacji wykonawczych (workers)
  - **GET_IMAGES_LIST** - serwer zwraca administorowi listę obrazów procesów zapisanych na serwerze
  - **UPLOAD_IMAGE** - administrator udostępnia obraz procesu na serwer - obraz zostaje zapisany na serwerze (wymaga wysłania obrazu procesu po wysłaniu tego komunikatu).
  - **DELETE_IMAGE** - administrator wywołując tę komendę może usunąć obraz o podanej nazwie z serwera (wymaga wysłania nazwy obrazu po wysłaniu tego komunikatu).
  - **RUN_NOW** - uruchomienie procesu z obrazu na workerze (nazwa obrazu procesu i ID workera wysyłane po tym komunikacie)
  - **STOP_NOW** - zatrzymanie wykonywania procesu na workerze (nazwa obrazu procesu i ID workera wysyłane po tym komunikacie)
  - **GET_WORKERS_IMAGES** - serwer zwraca administorowi listę obrazów procesów wysłanych do workera o identyfikatorze wysłanym po wysłaniu tego komunikatu
  - **UPLOAD_IMAGE_WORKER** - przesłanie obrazu z serwera do workera - wymaga wysłania nazwy obrazu oraz identyfikatora workera do którego obraz ma zostać wysłany
  - **DELETE_IMAGE_WORKER** - usunięcie obrazu z przestrzeni dyskowej workera - wymaga podania nazwy obrazu oraz identyfikatora workera
  - **RUN_SHEDULE** - zaplanowanie uruchomienia procesu z danego obrazu o danym czasie w ciągu doby na danym workerze
  - **RUN_SHEDULE** - zaplanowanie zatrzymania procesu z danego obrazu o danym czasie w ciągu doby na danym workerze
  - **CLEAR_SHEDULE** - wyczyszczenie planu uruchamiania i zatrzymowania procesu na danym workerze
  - **GET_SHEDULE** - pokazanie planu uruchamiania i zatrzymowania procesu na danym workerze
  - **CLOSE** - zamyka połączenie pomiędzy administratorem a serwerem
- Wysyłane z serwera do workera:

  - **GET_IMAGES_LIST** - zwraca listę obrazów wysłanych do danego workera.
  - **UPLOAD_IMAGE** - wysłanie obrazu z serwera do workera
  - **DELETE_IMAGE** - usunięcie obrazu z workera
  - **RUN_NOW** - uruchomienie procesu na workerze (nazwa obrazu wysyłana po tym komunikacie)
  - **STOP_NOW** - zatrzymanie obrazu na workerze (nazwa obrazu wysyłana po tym komunikacie)
  - **RUN_SHEDULE** - zaplanowanie uruchomienia procesu z danego obrazu o danym czasie w ciągu doby
  - **RUN_SHEDULE** - zaplanowanie zatrzymania procesu z danego obrazu o danym czasie w ciągu doby
  - **CLEAR_SHEDULE** - wyczyszczenie planu uruchamiania i zatrzymowania procesu
  - **GET_SHEDULE** - pokazanie planu uruchamiania i zatrzymowania procesu

- Wysyłane z workera do serwera:
  - **WORKER** - nawiązanie połączenia między workerem a serwerem
  - **GET_IMAGES_LIST_RESPONSE** - odpowiedź na komunikat **GET_IMAGES_LIST**
  - **UPLOAD_IMAGE_RESPONSE** - odpowiedź na komunikat **UPLOAD_IMAGE**
  - **DELETE_IMAGE_RESPONSE** - odpowiedź na komunikat **DELETE_IMAGE**
  - **RUN_NOW_RESPONSE** - odpowiedź na komunikat **RUN_NOW**
  - **STOP_NOW_RESPONSE** - odpowiedź na komunikat **STOP_NOW**
  - **RUN_SHEDULE_RESPONSE** - odpowiedź na **RUN_SHEDULE**
  - **RUN_SHEDULE_RESPONSE** - odpowiedź na **RUN_SHEDULE**
  - **CLEAR_SHEDULE_RESPONSE** -odpowiedź na **CLEAR_SHEDULE**
  - **GET_SHEDULE_RESPONSE** - odpowiedź na **GET_SHEDULE**

4 Wnioski z wykonanego testowania

Dzięki przeprowadzonemu testowaniu mamy pewność, że:

- komunikacja między administratorem, workerem a serwerem z oparciem gniazd BSD została zrealizowana poprawnie.
- w przypadku przerwania połączenia internetowego utracone zostaje połączenie pomiędzy workerem a serwerem.

- na podstawie wskazań w programie wireshark oraz porównaniu rozmiaru wysłanych danych, wyniki okazują się być zgodne z założeniami. Świadczy to o dobrej implementacji programu nasłuchującego, a także o poprawnym działaniu analizatora ruchu.

5 Instrukcja instalacji stworzonego systemu

Jako zespół do zbudowania projektu używaliśmy środowiska CLion. W takim przypadku wystarczy tylko kliknąć przycisk *"Run"* i projekt będzie gotowy do uruchomienia. Dane środowisko pomaga także w automatycznym uruchamianiu testów: *"run_acceptance_tests.py"*. Zaimplemetowane przez nas rozwiązanie równie dobrze można zbudować "ręcznie"poprzez wykonanie następującej sekwencji poleceń:

1. mkdir build
2. cd build
3. cmake ..
4. make

Po zbudowaniu zgodnie z instrukcją, należy przejść do katalogu
"*/DistributedProcesses/bin* "
i poleceniem "*/server -a* ***[tu należy podać adres ip]*** "uruchomić serwer.
Admin i worker uruchamiają się analogicznie. W razie wątpliwości można dodać flagę ***-h***, sprawi ona, że w oknie terminala pojawią się wskazówki dotyczące dostępnych trybów działania programu oraz możliwych opcji do wykorzystania.

6 Podsumowanie

6.1 Wyciągnięte doświadczenia

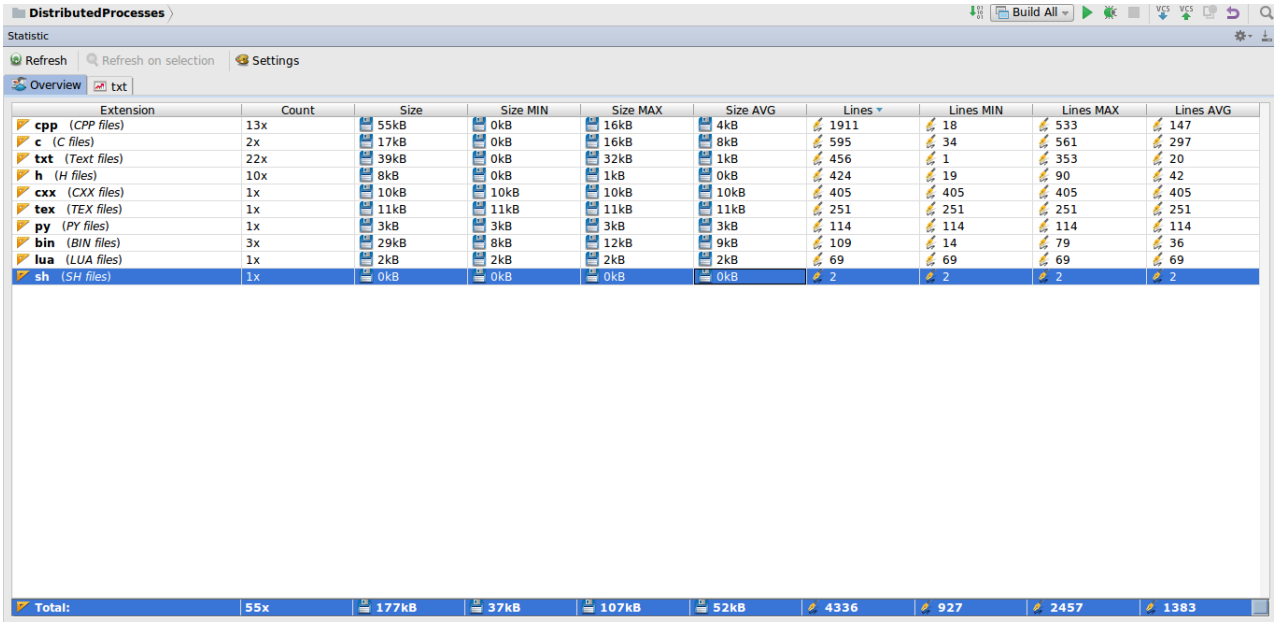
Komunikacja sieciowa. Przede wszystkim stworzony system pozwolił nam nauczyć się wielu zagadnień w zakresie komunikacji sieciowej. Po raz kolejny upewniliśmy się jak ważne są testy jednostkowe oraz integracyjne. Mieliśmy możliwość zapoznać się z tworzeniem skryptów LUA. Dany projekt dał nam dodatkową możliwość poćwiczyć techniki pisania czystego i działającego kodu.

Github. Bez systemu kontroli wersji **github** realizacja projektu byłaby niemożliwa. Tworzenie zadanego systemu jest czasochłonne, powstaje przy tym dużo różnych wersji programu. Warto, żeby każdy członek zespołu od samego początku w szczegółach zapoznał się ze strukturą zdalnego repozytorium oraz jego obsługą. W przypadku naszego zespołu oprócz kontroli wersji i code review, została wykorzystana również funkcja Issues do przydziału przez Marcina poszczególnych zadań członkom zespołu. Pozwoliło to na bieżąco śledzić proces wykonania przydzielonych zadań.

Umiejętności miękkie. Kolejnym punktem, który warto wymienić jest dobra dokumentacja wstępna oraz w miarę równomierny podział obowiązków. W przypadku odłożenia projektu na czas późniejszy, jego wykonanie nie byłoby możliwe jak również jeśliby cały projekt miała zrealizować jedna osoba. Tworząc coś wspólnego nauczyliśmy się współpracować ze sobą. Marcin sprawdził się jako lider zespołu.

6.2 Rozmiar plików

Dokładne statystyki dotyczące rozmiarów plików udało się uzyskać przy pomocy wtyczki Statistic w środowisku CLion (produkcja JBrains).



Rysunek 4: Statystyki rozmiaru projektu

6.3 Oszacowanie czasu pracy

Tablica 3: Oszacowanie liczby godzin pracy nad projektem

CEL	LICZBA GODZIN
Zapoznanie się z dokumentacją techniczną BSD	20
Projektowanie systemu	30
Implementacja	100
Testowanie	30
Naprawa błędów	50
Razem	ok. 230