# SCZR - dokumentacja końcowa
# Grupa nr 28

Krystian Kujawski (291129), Łukasz Pokorzyński (300251), Adam Steciuk (300263)
Styczeń 2021

## 1 Wykorzystane dodatkowe narzędzia

- Python z bibliotekami MatPlotLib i NumPy
- Stress - narzędzie do generowania sztucznego obciążenia procesora [LINK]

## 2 Opis systemu i jego funkcjonalności

System przez nas zaprojektowany jest symulacją systemu zbierającego dane (temperaturę) z pewnej liczby pomniejszych stacji meteorologicznych. Liczba symulowanych stacji jest ustalana przez użytkownika korzystającego z systemu przed rozpoczęciem symulacji. Użytkownik ustala też tryb szeregowania procesów oraz jeżeli wybrany tryb planisty jest dostosowany pod systemy czasu rzeczywistego, ustala również, czy proces wizualizujący powinien być powiązany z rdzeniem procesora. Pozyskane informacje system przenosi na obraz aktualizowany w czasie rzeczywistym obszarowo wyświetlający różnice temperatur. System został przygotowany z myślą o stacji roboczej działającej pod systemem Linux wyposażonej w monitor.
System składa się z 4 procesów działających w czasie rzeczywistym:

- **Proces A** - generacja danych z wirtualnych stacji meteorologicznych
- **Proces B** - zbieranie danych z symulowanych stacji i przekazywanie ich do procesu C
- **Proces C** - wizualizuje zebrane dane w postaci mapy izotermicznej z widocznym położeniem stacji
- **Proces D** - interfejs użytkownika, zbieranie danych diagnostycznych, które można zebrać w histogram przy pomocy skryptu

System został zrealizowany w języku C++. Skrypt do histogramów został napisany w języku Python i jest uruchamiany niezależnie od systemu. Do wizualizacji danych została wykorzystana biblioteka Allegro.
Oryginalnie planowaną biblioteką do wykorzystania była biblioteka OpenCV, lecz wybrany przez nas zestaw narzędzi zawiera wszystkie potrzebne funkcje do efektywnego wyświetlania wyników działania.

Dane diagnostyczne, przekazywane do systemu zarządzającego, to: ID stacji meteorologicznej, temperatura odczytana na stacji, czas trwania przekazania danych (proces B) lub czas od wysłania danych do ich wizualizacji (proces C).

Ogólny schemat prezentujący działanie systemu został ukazany niżej. Pokazuje on sposób komunikacji między procesami oraz przekazywane dane. Na pomarańczowo zostały ukazane kolejki komunikatów, kolor niebieski został przypisany pamięci współdzielonej. Kolorem żółtym zostały wyróżnione poszczególne procesy biorące udział w systemie. Dzięki zbieraniu danych diagnostycznych przez proces D można łatwo ustalić czas działania poszczególnych operacji krytycznych dla działania systemu.

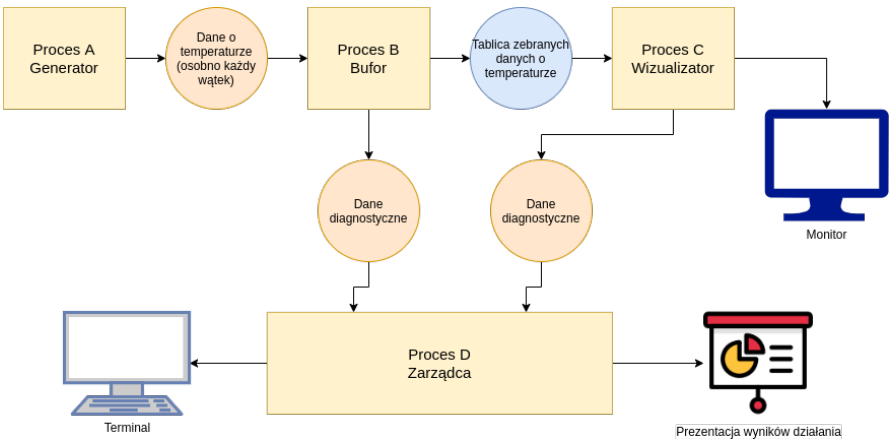



Diagram prezentujący architekturę systemu.

# 3 Opis elementów składowych systemu

## 3.1 Proces A

**Proces generujący dane**. Proces losowo generuje położenie zadanej liczby stacji na siatce o podanych wymiarach oraz początkową temperaturę dla każdej z nich. Następnie dla każdej ze stacji uruchamiany jest osobny wątek obsługujący zmiany temperatur. Do każdego wątku przekazywane są: id stacji, jej x-owa współrzędna, y-owa współrzędna, początkowa temperatura, maksymalny krok o jaki może zmienić się temperatura w stacji w każdej iteracji, minimalna i maksymalna osiągalna temperatura oraz wskaźnik do kolejki komunikatów.

Symulacja zmian temperatury stacji polega na dodaniu do obecnej temperatury losowo wybranej wartości z przedziału <-step, step>.

Zrezygnowaliśmy z podawania interwałów czasowych między generacjami temperatur. Kolejne generowane są tylko jak zwolni się miejsce w kolejce wysyłającej dane do procesu B.

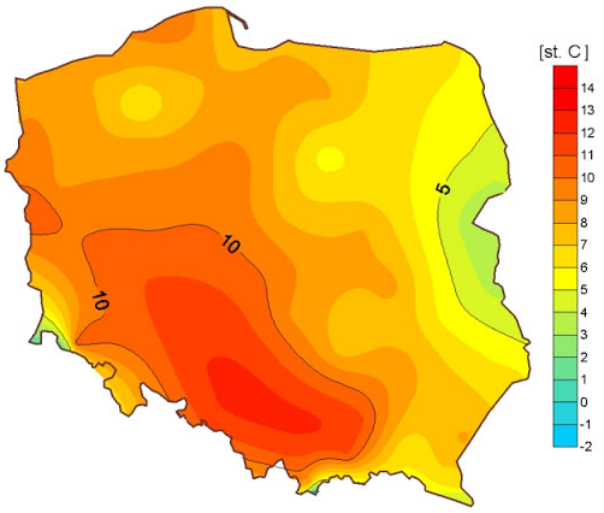
## 3.2 Proces B

**Proces buforujący**. Jest to proces, którego zadaniem jest zbieranie danych w specjalnie przygotowanej tablicy, a następnie, gdy tablica zbierze już pierwsze informacje ze wszystkich stacji, przekazuje je każdorazowo do procesu C (czyli przy każdej zmianie wartości odczytanej temperatury z danej stacji). Oryginalnie jego zadaniem miała być aproksymacja przebiegu wartości między stacjami w linii prostej, lecz zadanie wyznaczania rozkładu temperatury na mapie zostało przekazane procesowi C. W tym celu proces B przekazuje przy pomocy pamięci współdzielonej ID stacji, która ostatnio przekazała swoje pomiary (wraz z tymi pomiarami), tablicę ze współrzędnymi symulowanych stacji oraz ich ostatnimi pomiarami.

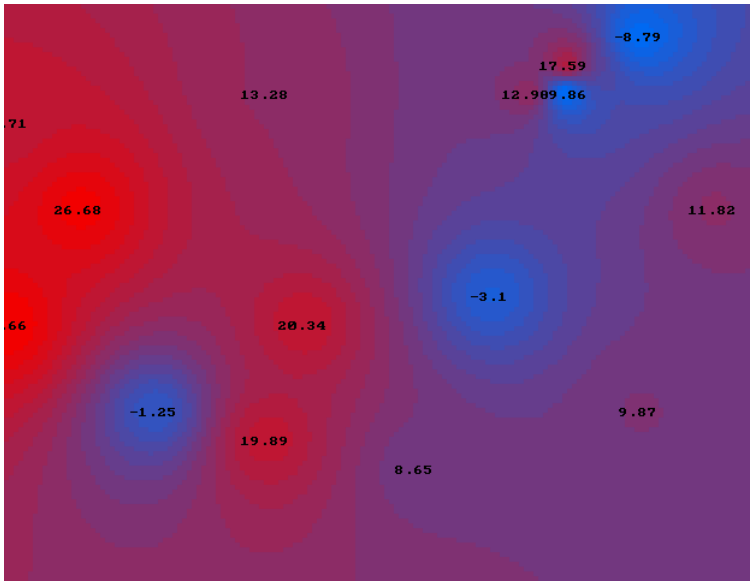
Poza zbieraniem danych z symulowanych stacji, proces ten jest odpowiedzialny za przekazywanie danych diagnostycznych do procesu zarządcy. Diagnostyka ta ma wskazać, ile czasu mija od przekazania danych przez proces A do kolejki komunikatów, do ich odebrania przez proces B.

## 3.3 Proces C

**Proces wizualizujący**. Proces otrzymuje od procesu B tablicę z danymi o umiejscowieniu stacji oraz o odczytanej w nich temperaturze. Wyszukuje on maksymalną oraz minimalną odczytaną temperaturę, aby móc je przedstawić odpowiednimi kolorami. Dla każdego piksela w wyświetlanym oknie obliczana jest odległość od każdej ze stacji, na podstawie których, program wyznacza (korzystając ze średniej ważonej po tych odległościach) temperaturę w punkcie reprezentowanym przez dany piksel. Po każdorazowym otrzymaniu danych od procesu B tworzy on nową zaktualizowaną mapę. Po utworzeniu mapy izotermicznej nakłada otrzymane temperatury ze stacji na ich lokalizację, aby użytkownik mógł łatwiej zinterpretować otrzymany wynik. Równocześnie proces przekazuje informacje do procesu D dotyczące ID stacji, temperatury odczytanej w tej stacji oraz czasu, jaki minął od otrzymania danych do utworzenia mapy izotermicznej.




Zamierzony efekt




Obraz wyświetlany przez proces C

### 3.4 Proces D

**Zarządca i interfejs użytkownika**. Proces odpowiedzialny za rozruch całego systemu symulacyjnego. W trakcie jego uruchamiania tworzone są semafory, pamięć współdzielona i otwierane kolejki do obsługi komunikatów i przekazywanych danych. Pozwala on na ustalenie strategii szeregowania procesów. Jeżeli szeregowanie, które zostało wybrane, jest szeregowaniem dla czasu rzeczywistego, to dodatkowo pyta o chęć związania procesu wizualizującego z wybranym rdzeniem procesora. Z procesu ustalana jest liczba symulowanych stacji meteorologicznych oraz rozdzielczość generowanej mapy (x na y możliwych stacji do generacji). Możliwość utworzenia histogramów z przekazanych danych diagnostycznych została odseparowana od procesu i dokonuje się tego przez osobny skrypt w języku Python przy wykorzystaniu bibliotek MatPlotLib i NumPy.

### 3.5 Kanały komunikacyjne

Kanały komunikacje w zbudowanym systemie zostały zrealizowane przy pomocy rozwiązań POSIX dostępnych w systemie - kolejek komunikatów, pamięci współdzielonej oraz semaforów. Zapewniają one sprawne powiadamianie procesów o dostępności danych jak i synchronizacji działań poszczególnych procesów. Oryginalnie planowaliśmy do głównej komunikacji A-B i B-C użyć pamięci współdzielonej razem z kolejkami komunikatów, lecz odpowiednio została użyta pojedyncza kolejka komunikatów dla kanału A-B oraz pamięć współdzielona z semaforami dla kanału B-C. W związku z tymi rozwiązaniami bufor cykliczny nie został wykorzystany przy implementacji pamięci współdzielonej.

**Kanał A-B**. Proces A, składający się z kilku wątków reprezentujących pojedyncze stacje meteorologiczne, jest obsługiwany przez kolejkę komunikatów. Dzięki swojej wbudowanej możliwości blokowania procesów piszących do/czytających z kolejki w razie przepełnienia (bądź braku komunikatów w przypadku czytających), kolejka zapewnia prawidłową synchronizację między wieloma wątkami. Jako, że wszystkie procesy mają ten sam priorytet, to z kolejki zawsze jest wyciągana wiadomość najstarsza.

**Kanał B-C**. Zebrane wartości w specjalnej strukturze są przekazywane przez proces B w pamięci współdzielonej, z której czyta proces C. Poprawna synchronizacja przesłania i odebrania danych jest kontrolowana przez zdefiniowane do tego celu semafory. Inny sposób synchronizacji nie był potrzebny, ponieważ jedynie proces B (w roli producenta) i proces C (konsument) maja dostęp do pamięci.

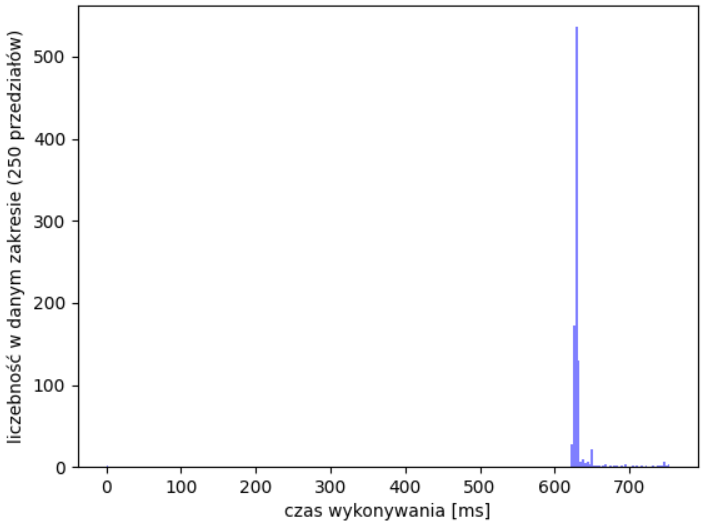
**Kanał B-D oraz C-D**. Kanały przesyłają dane diagnostyczne do procesu D, które później są wykorzystywane do analizy zbudowanego systemu. Kanał B-D przesyła ID stacji, od której przyszły najświeższe informacje, przekazaną przez stację temperaturę oraz czas od wysłania danych przez jeden z wątków procesu A do ich odebrania w procesie B. Kanał C-D tak samo jak kanał B-D wysyła ID i temperaturę oraz czas, lecz jest on mierzony od momentu wysłania danych przez proces B, do momentu wygenerowania mapy.

# 4 Testy i wnioski

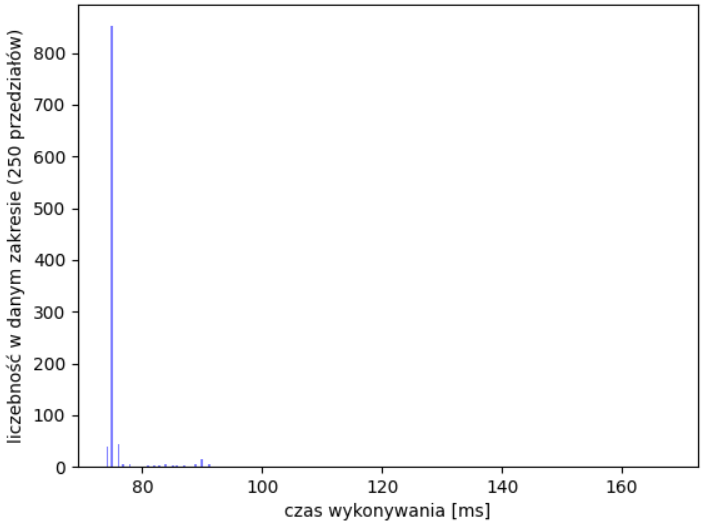
Wykresy zbudowane zostały na podstawie 1000 zgromadzonych logów (przypadki 15 stacji), 500 logów (przypadki 50 stacji) oraz około 100 (dla jednego przypadku 100 stacji). Dla przypadków 15 i 50 stacji możemy zauważyć, że działanie zbudowanego systemu jest stabilne, normuje się w okolicach pewnych powtarzających się wartości. Dla przypadków 50 symulowanych stacji możemy zauważyć znaczny skok w czasie rejestrowanym zarówno dla logów pochodzących z procesu B, jak i C. Przez większą liczbę stacji, a co za tym idzie większą liczbę wątków pracujących w obrębie procesu A, liczba oczekujących stacji do kolejki staje się znacznie większa, co wpływa na długość rejestrowanego czasu od rozpoczęcia operacji push do kolejki komunikatów do odebrania danych niesionych z tym komunikatem.

Jeszcze ciekawszą sytuację możemy zaobserwować dla przypadku domyślnego planisty z setką stacji - tutaj liczba wątków jest już na tyle duża, że powoduje brak stabilizacji czasów wokół pewnej wartości i coraz dłuższe czasy oczekiwania do kolejki komunikatów, a czas rysowania mapy (log_c) wzrasta prawie 20-krotnie w stosunku do przypadku z 15 stacjami.

**Domyślny planista, 15 stacji**

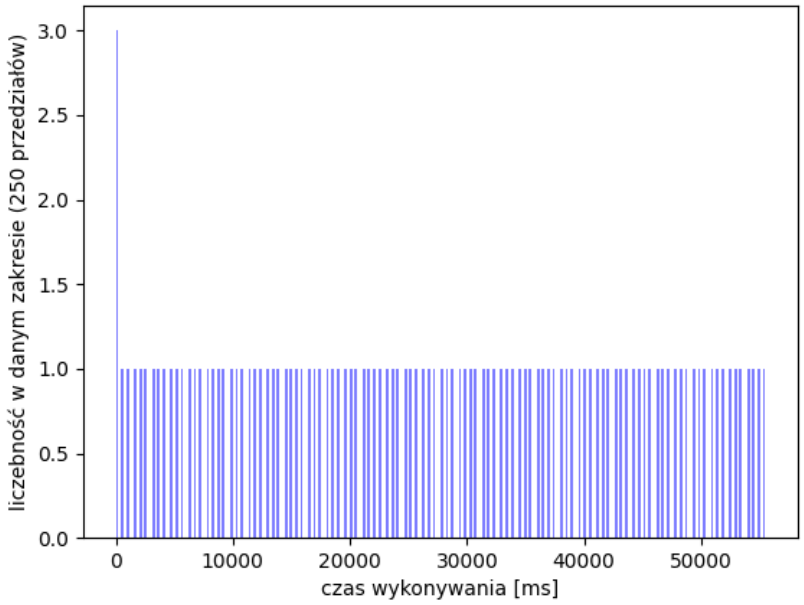



log_b

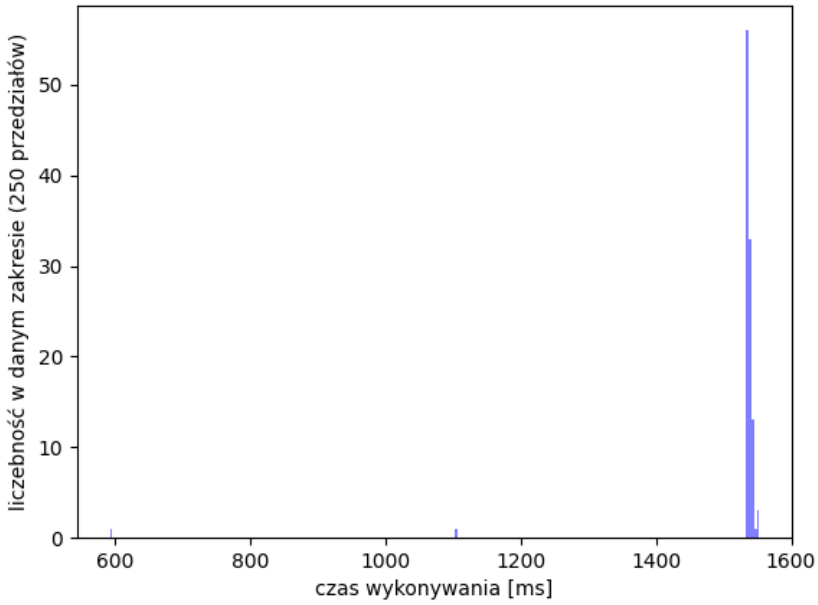



log_c

**Domyślny planista, 100 stacji**.

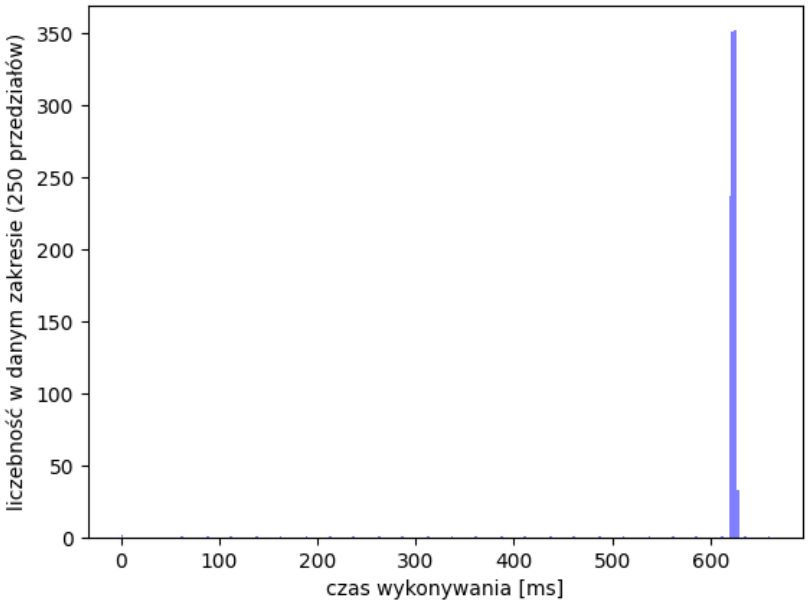



log_b

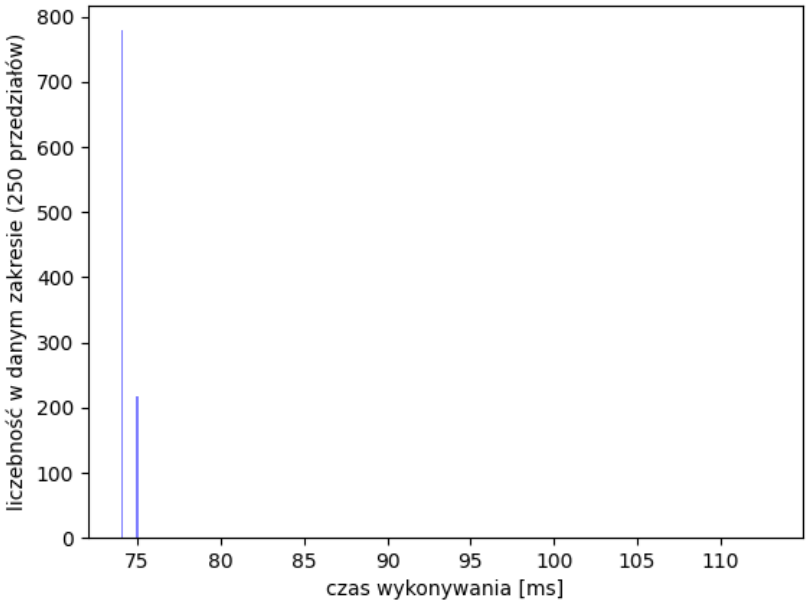



log_c

**Planista FIFO, 15 stacji**.

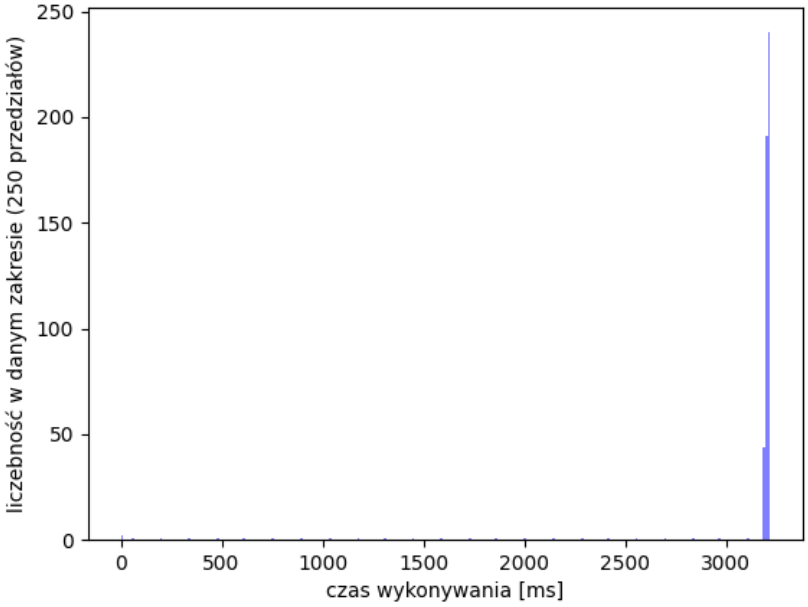



log_b

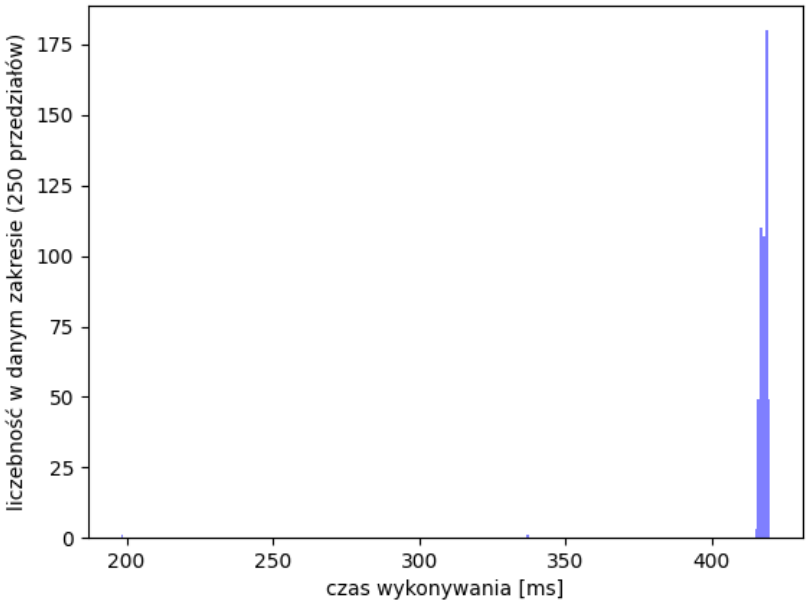



log_c

**Planista FIFO, 50 stacji**.

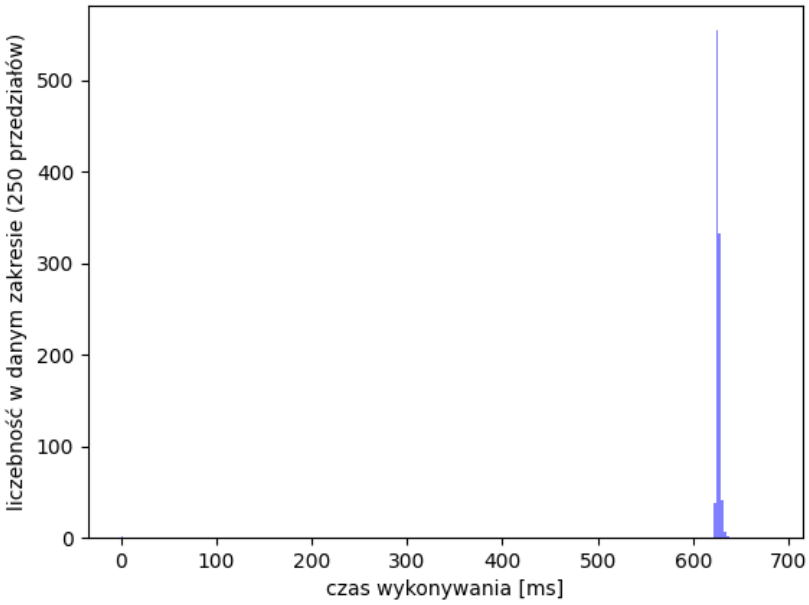



log_b

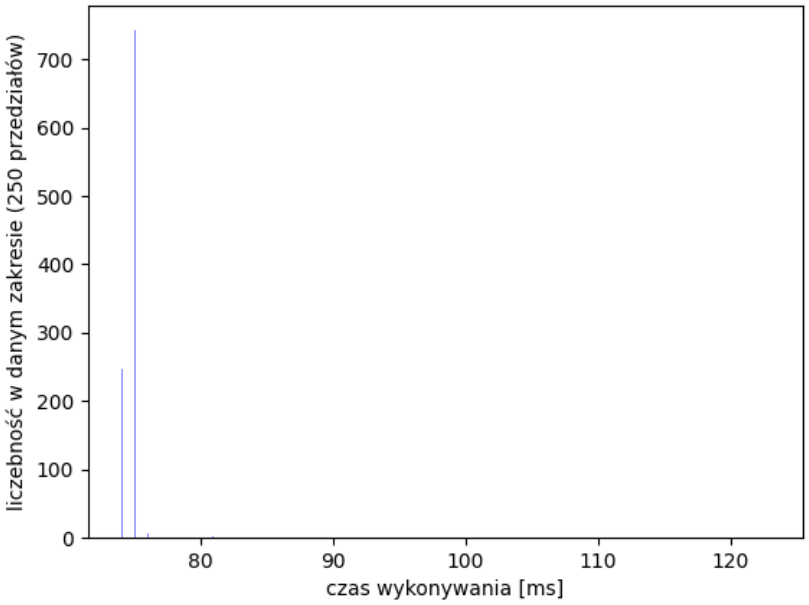



log_c

**Planista RR, 15 stacji**.

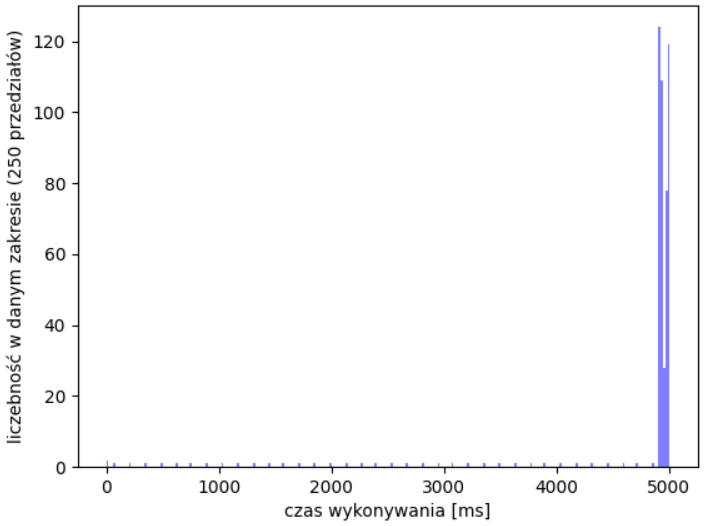



log_b

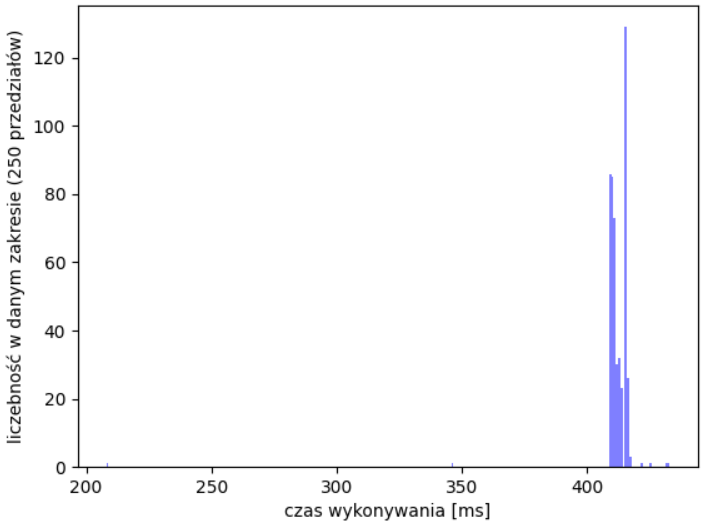



log_c

**Planista RR, 50 stacji**.



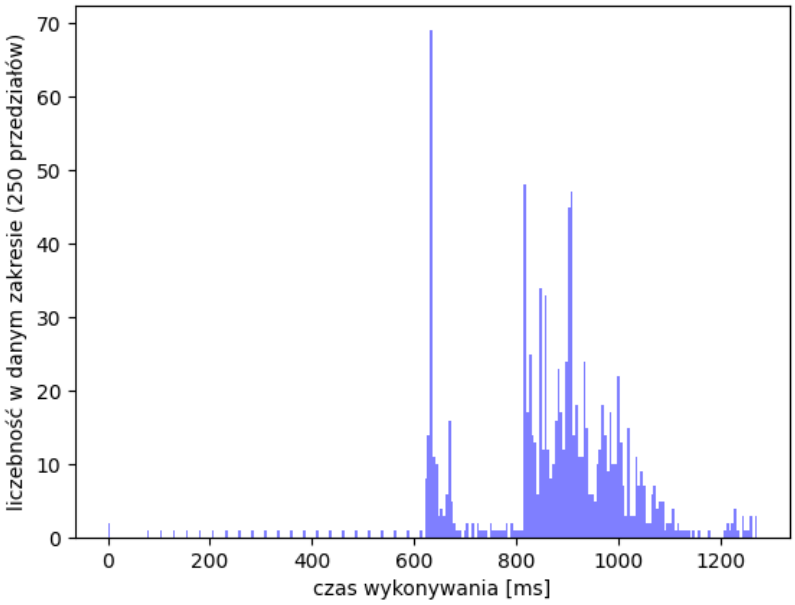

log_b



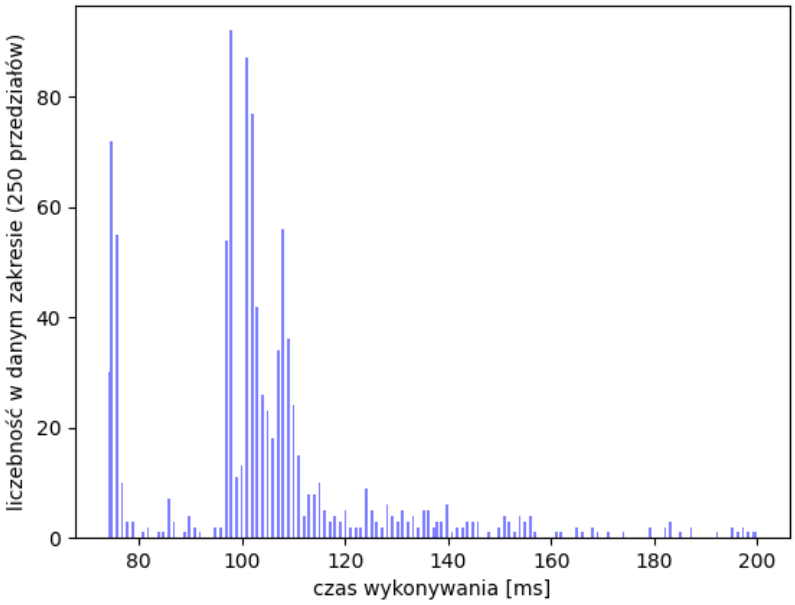

log_c

Testy obciążeniowe przeprowadziliśmy z pomocą programu "*stress*", który pozwala na symulację obciążenia procesora stacji roboczej i innych podzespołów odpowiedzialnych za zasoby sprzętowe. Przy ich okazji mogliśmy zauważyć większą wydajność programu przy uruchomieniach pod obciążeniem w szeregowaniu FIFO/RR niż w szeregowaniu domyślnym. Przejawia się to m.in. w większej stabilności czasów, co możemy zobaczyć na wykresach. Powodem tego jest fakt, że procesy uruchamiane w polityce FIFO/RR zwalniają proces albo dobrowolnie albo gdy zostają wywłaszczone przez inne procesy czasu rzeczywistego. Procesy programu "*stress*" nie są procesami RT więc obciążenie w tym przypadku w znacznie mniejszym stopniu wpływa na wydajność naszego programu.

**Domyślny planista, 15 stacji z obciążeniem**.

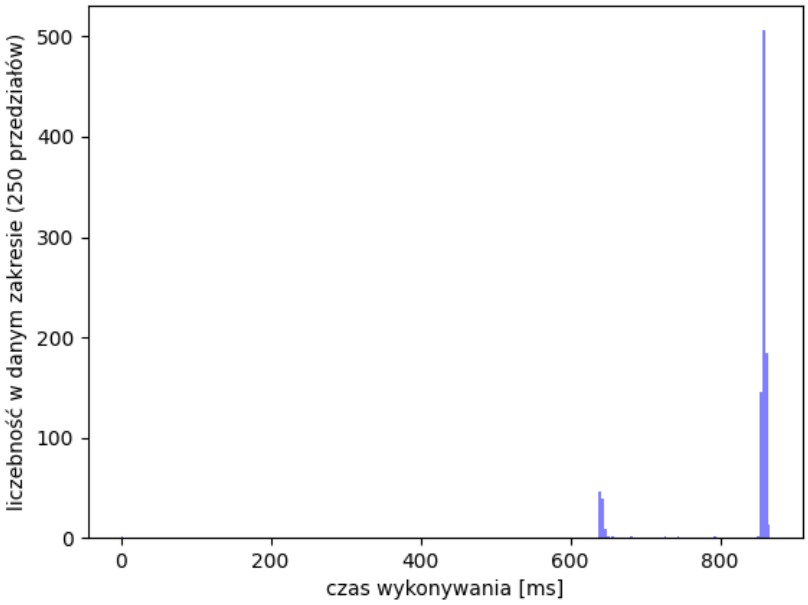



log_b

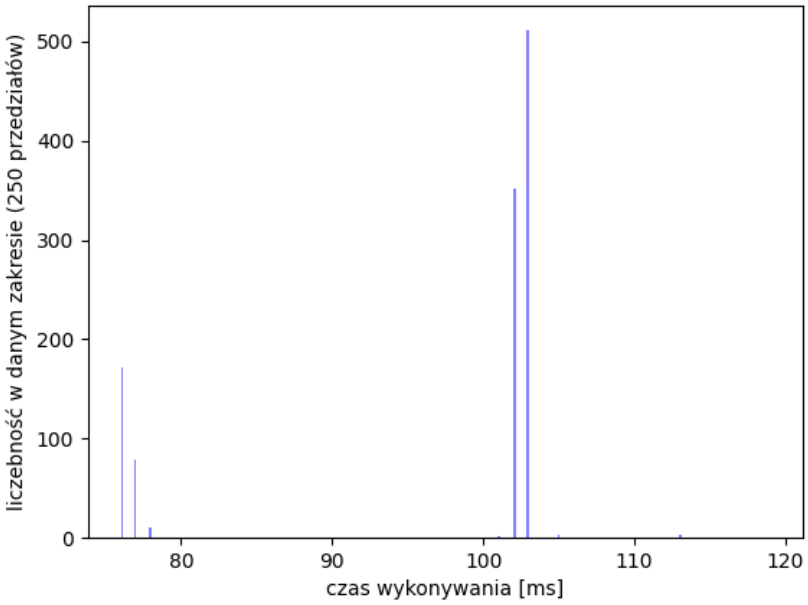



log_c

**Planista FIFO, 15 stacji z obciążeniem**.

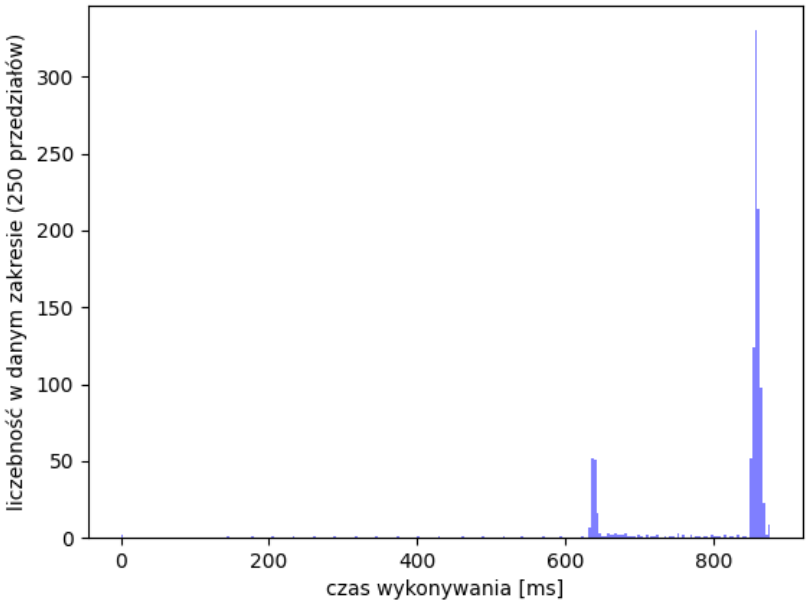



log_b

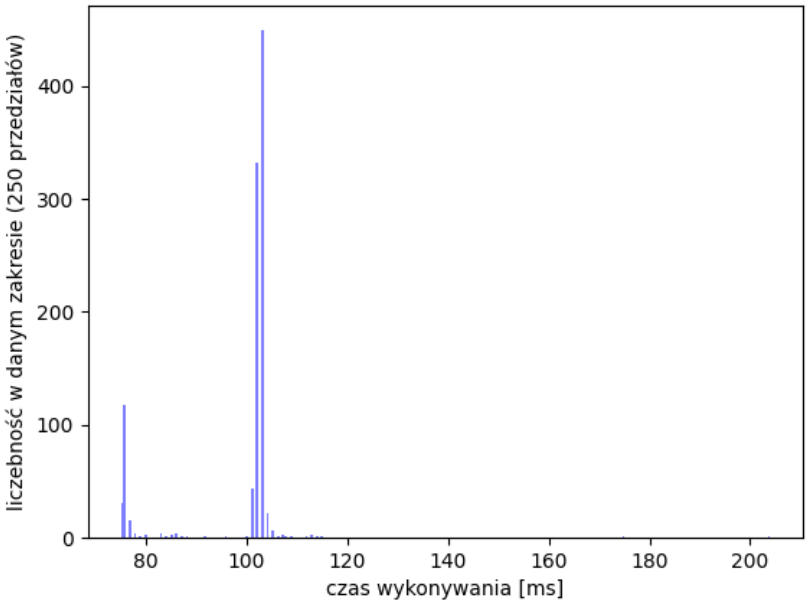



log_c

**Planista RR, 15 stacji z obciążeniem**.




log_b




log_c

## Podział pracy

- **Krystian Kujawski** - konfiguracja i obsługa biblioteki graficznej (proces C)
- **Łukasz Pokorzyński** - projekt interfejsu użytkownika, architektura systemu, uruchamianie systemu (Proces D)
- **Adam Steciuk** - implementacja symulacji stacji meteorologicznych i generowanych przez nie danych (proces A), obliczanie gradientów dla mapy wynikowej.

Przygotowanie dokumentacji końcowej oraz przeprowadzenie testów leżało w obowiązku wszystkich członków zespołu.

## Czego się nauczyliśmy

Przede wszystkim opanowaliśmy podstawy rozwiązań wielowątkowych i wieloprocesowych. Poszerzyliśmy naszą wiedzę na temat sposobów implementacji semaforów, pamięci współdzielonej, kolejek komunikatów oraz systemowego szeregowania procesów. Dodatkowo poznaliśmy składnie i sposób w jaki można używać biblioteki graficznej Allegro5. Pomoże nam to w przyszłości jeszcze płynniej niż wcześniej pokonywać problemy związane z implementacją podobnych rozwiązań w innych bibliotekach i nie tylko.