Jakub Rozkosz, Hubert Gołębiowski

SKPS lab4 - komunikacja międzyprocesowa
Sprawozdanie

1. Przetestowanie działania programów na “gospodarzu”

Programy cw4a i cw4b pobrano z Moodla i zbudowano je poleceniem *make*.
Ustawiono właściwą ścieżkę do plików źródłowych komendą *export PATH=$PATH:*

Program działa poprawnie, generowane są pliki server.txt oraz cli_X.txt.

2. Zbudowanie pakietu dla OpenWRT

Usunięto wyniki poprzedniego budowania (na “gospodarzu”) poleceniem `make clean.`
Pobrano i rozpakowano SDK OpenWRT.
Następnie skompilowano pakiet przy pomocy SDK OpenWRT i zainstalowano go na RPi.

3. Ustalenie granicznej wartości czasu przetwarzania

1. Pierwszy wariant: 3 klientów, 1 rdzeń, pełne obciążenie

W pliku /boot/user/cmdline.txt dopisano *maxcpus=1* i zrestartowano RPi.
Pełne obciążenie osiągnięto uruchamiając w tle proces: *stress-ng --matrix 0 -t 1m*
Uruchomiono program z podanymi parametrami.
stress-ng --matrix 0 -t 5s & cw4a 3 50 10000 X
gdzie X zmieniano.
Ustalono wartość graniczną na: 250000

2. Drugi wariant: 3 klientów, 2 rdzenie, pełne obciążenie

W pliku zmieniono: maxcpus=2.
Uruchomiono program:
stress-ng --matrix 0 -t 5s & cw4a 3 50 10000 X
Ustalono wartość graniczną na: 380000

3. Trzeci wariant: 3 klientów, 2 rdzenie, bez obciążenia

Uruchomiono program bez obciążenia:
cw4a 3 50 10000 X
Ustalono wartość graniczną na: 520000

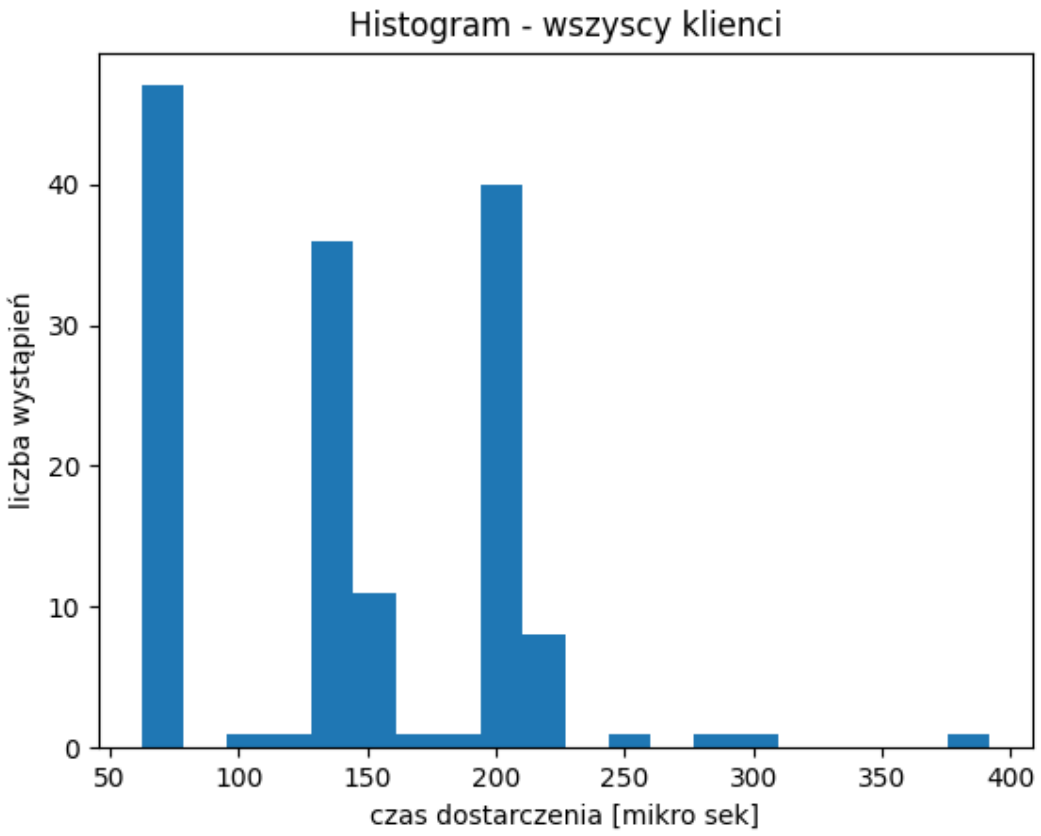
4. Czwarty wariant: 1 klient, 4 rdzenie, bez obciążenia

Ustawiono 4 rdzenie i uruchomiono program:
cw4a 1 50 10000 X
Ustalono wartość graniczną na: 790000

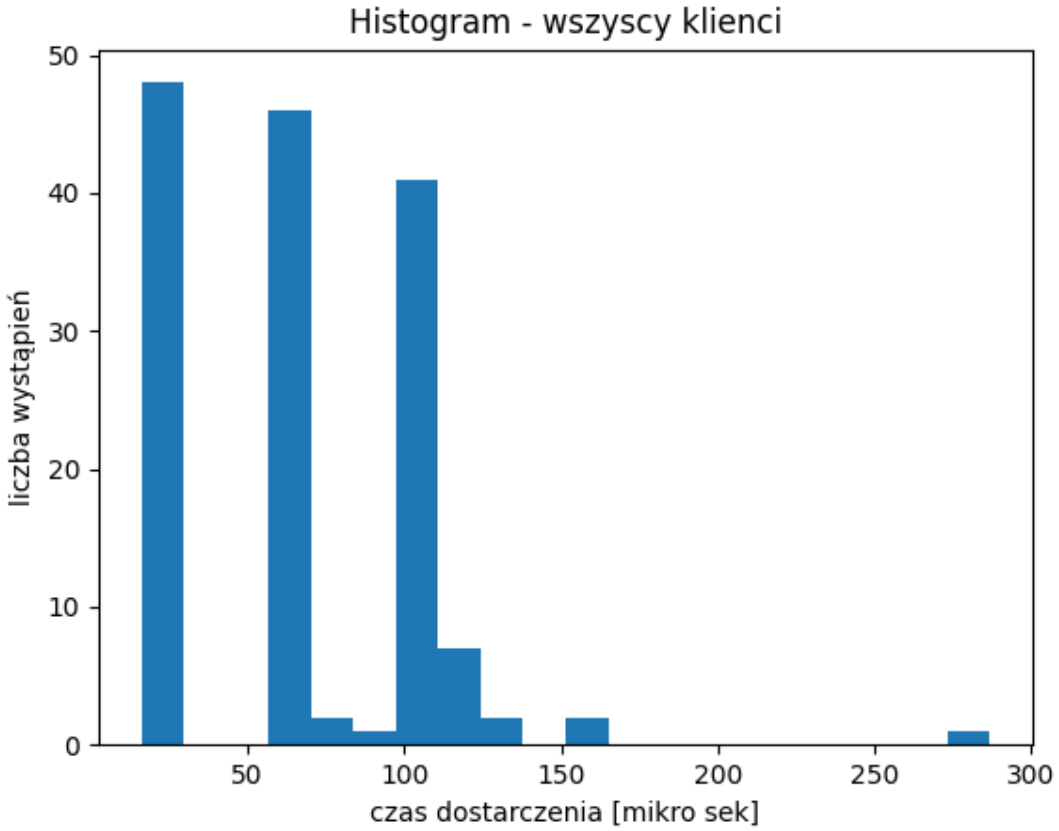
4. Rozkład czasu dostarczenia danych

W każdym przypadku na podstawie zebranych danych z plikow server.txt oraz cli_X.txt rysowano histogramy w pythonie korzystając z matplotlib.

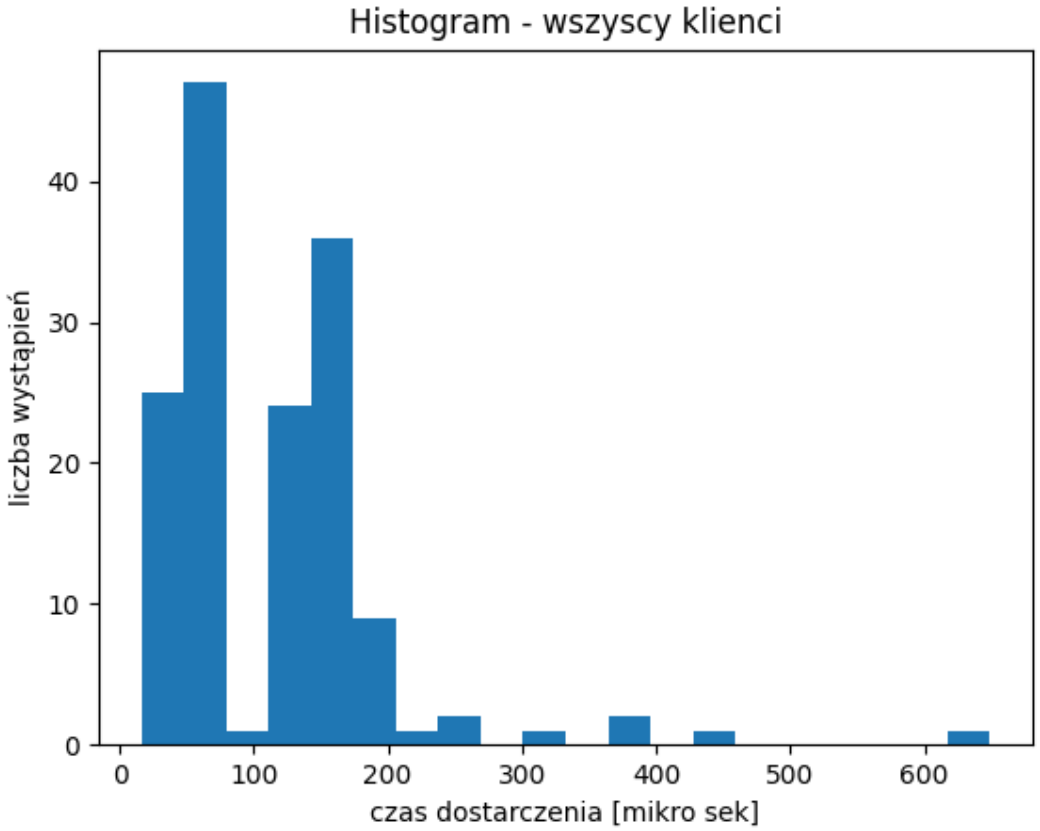
1. Pierwszy wariant: 3 klientów, 1 rdzeń, pełne obciążenie
Połowa wartości granicznej czasu przetwarzania: 125000



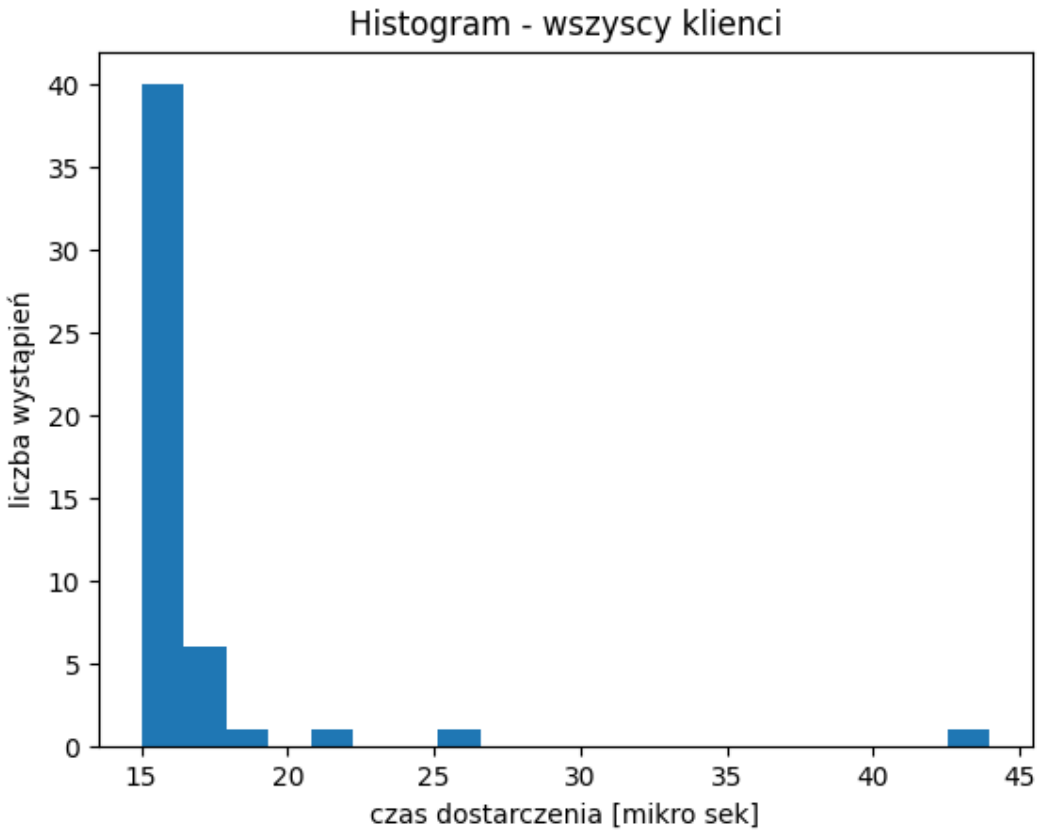
2. Drugi wariant: 3 klientów, 2 rdzenie, pełne obciążenie
Połowa wartości granicznej czasu przetwarzania: 190000



3. Trzeci wariant: 3 klientów, 2 rdzenie, bez obciążenia
Połowa wartości granicznej czasu przetwarzania: 260000



4. Czwarty wariant: 1 klient, 4 rdzenie, bez obciążenia
Połowa wartości granicznej czasu przetwarzania: 395000



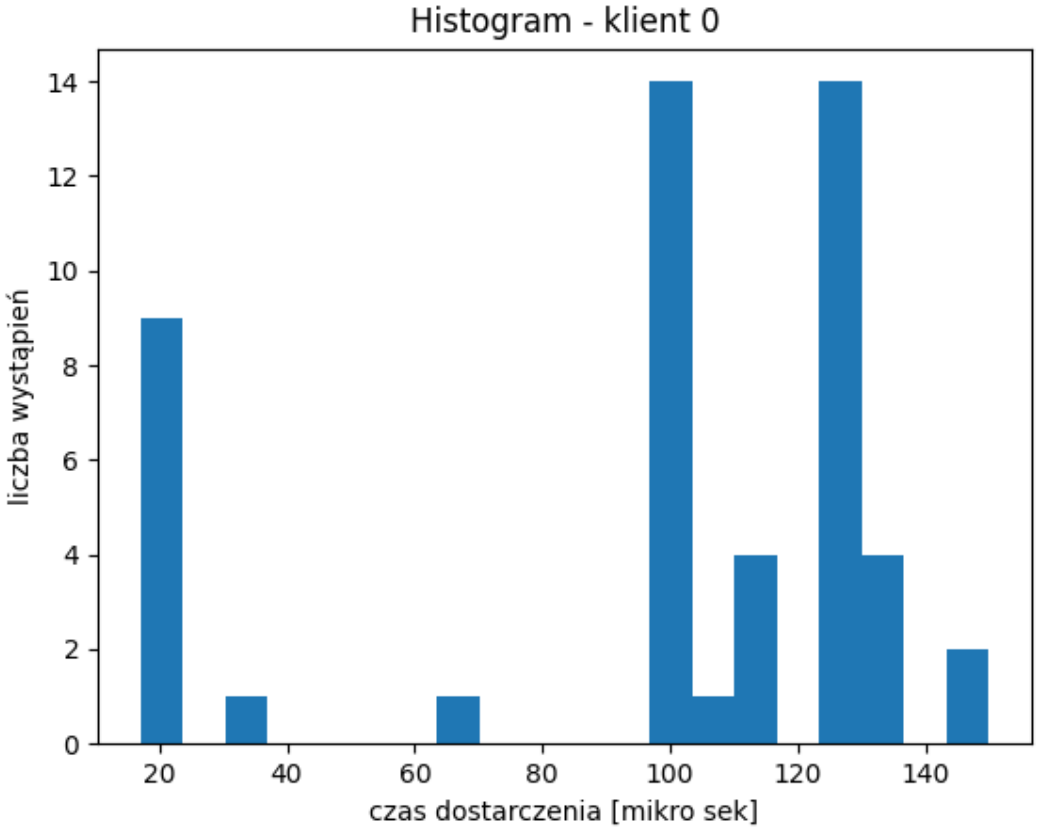
5. Aktywne oczekiwanie

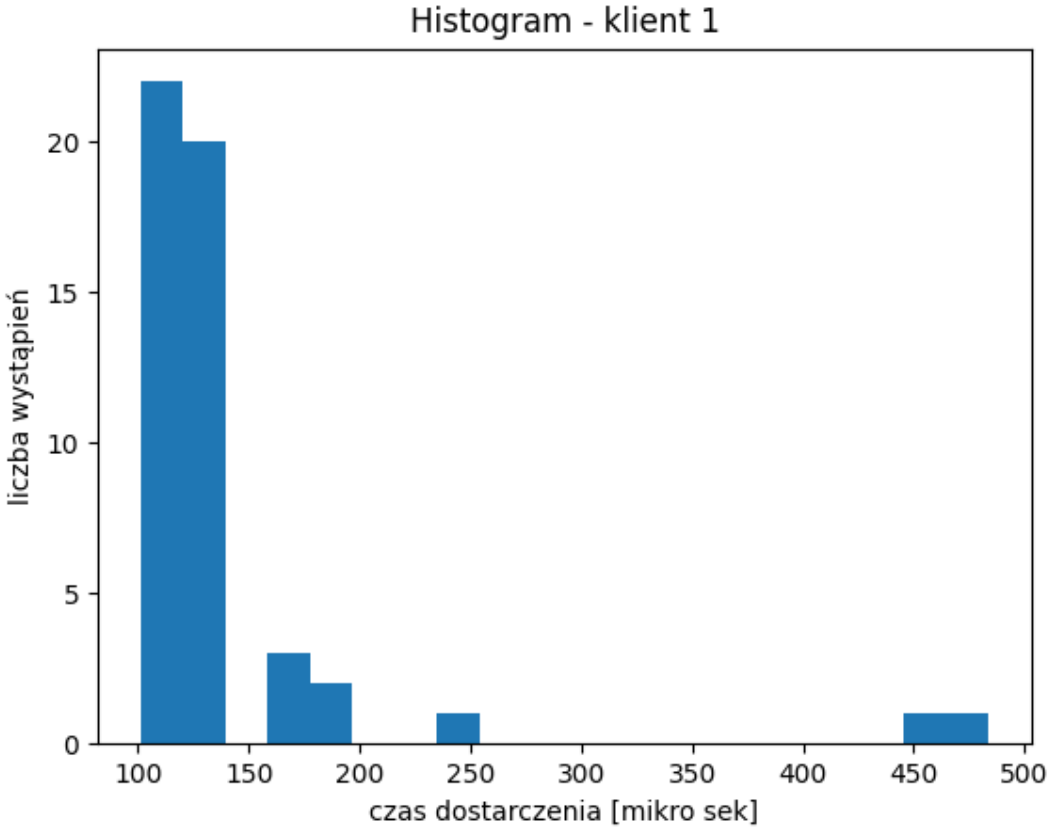
Program uruchamiano dla następujących parametrów:
2 klientów, 2 rdzenie, bez obciążenia

Modyfikacja aplikacji klienckiej cw4b:

1. klient 0 oczekuje aktywnie, reszta usypiana

```
79  79          } else {
80      -           pthread_cond_wait(&rbuf->cvar,&rbuf->cvar_lock);
    80  +           if (ncli != 0) {
    81  +               pthread_cond_wait(&rbuf->cvar,&rbuf->cvar_lock);
    82  +           }
81  83              pthread_mutex_unlock(&rbuf->cvar_lock);
82  84          }
```


Histogram - klient 0
liczba wystąpień
0
2
4
6
8
10
12
14
20
40
60
80
100
120
140
czas dostarczenia [mikro sek]

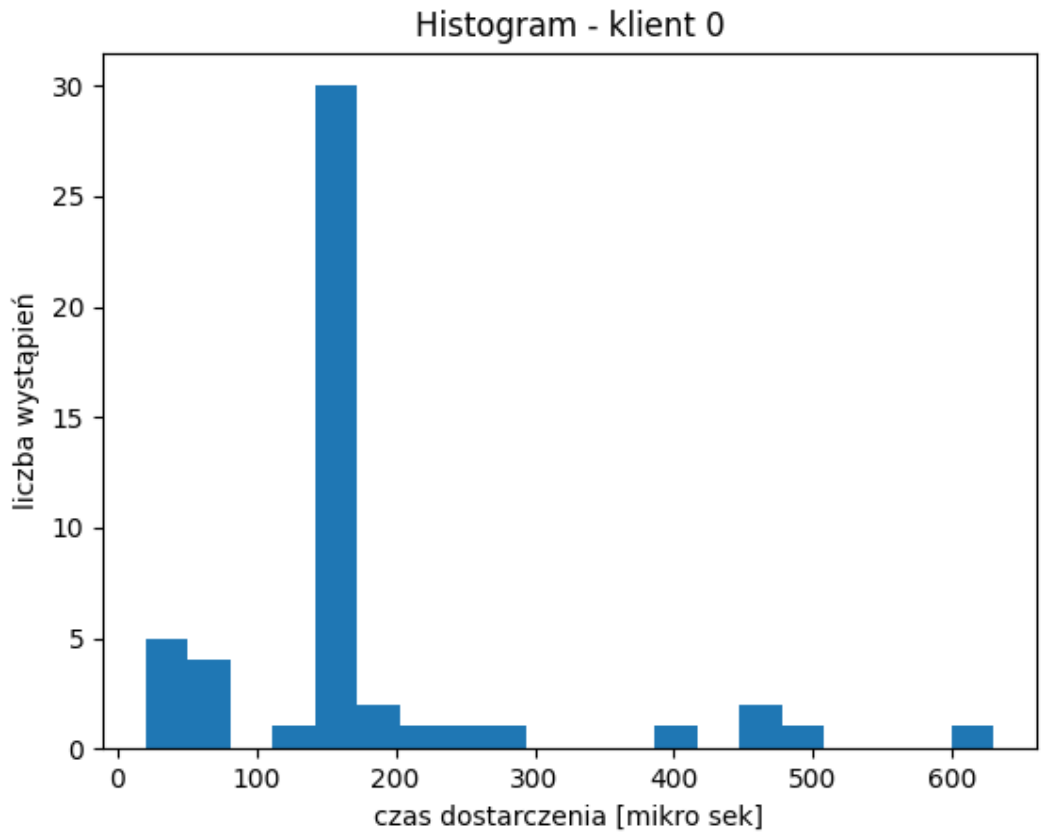

Histogram - klient 1
liczba wystąpień
0
5
10
15
20
100
150
200
250
300
350
400
450
500
czas dostarczenia [mikro sek]

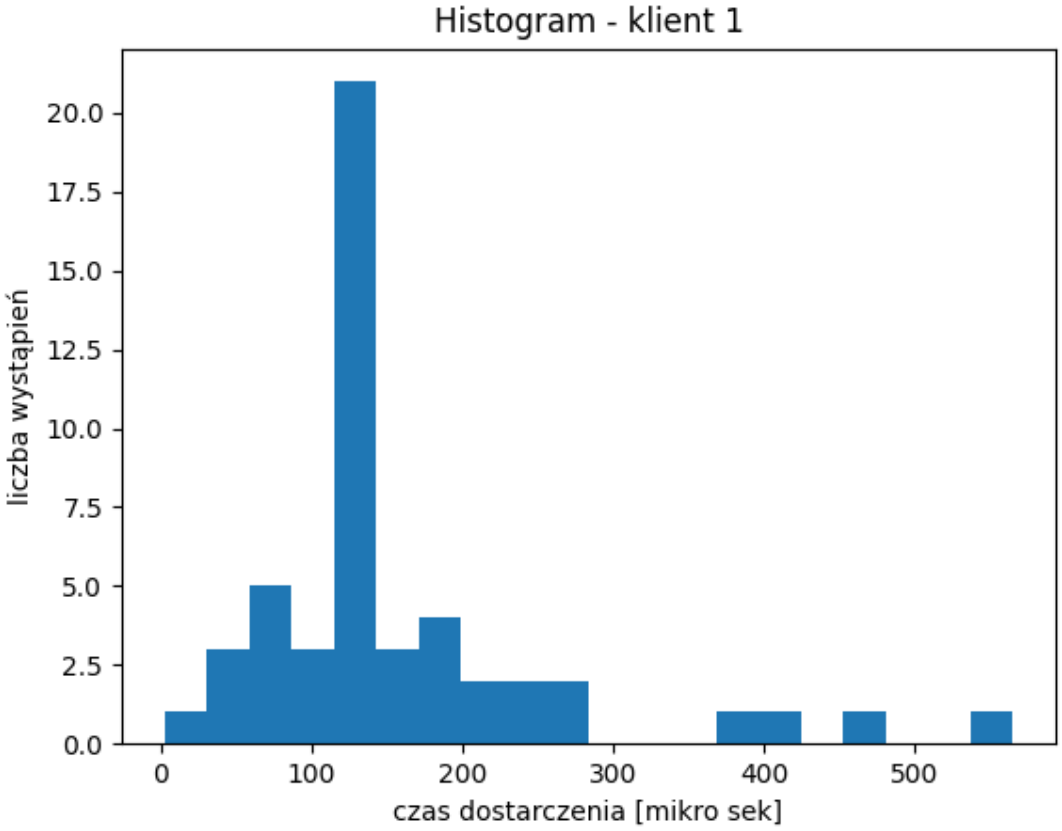
WNIOSKI:

Możemy zobaczyć, że oczekiwanie klienta 0 jest zazwyczaj krótsze niż oczekiwanie klienta 1. Jest to spowodowane tym, że klient 0 aktywnie oczekuje, więc ma szybsza rekację na przydzielenie procesora niż klient 1, który jest usypiany.

2. wszyscy klienci aktywnie oczekują

```
...  ...  @@ -77,7 +77,7 @@ int main(int argc, char *argv[])
77   77           for(j=0;j<ndel;j++)
78   78              dummy++;
79   79       } else {
80      -        pthread_cond_wait(&rbuf->cvar,&rbuf->cvar_lock);
     80   +        // pthread_cond_wait(&rbuf->cvar,&rbuf->cvar_lock);
81   81           pthread_mutex_unlock(&rbuf->cvar_lock);
82   82       }
83   83   }
```



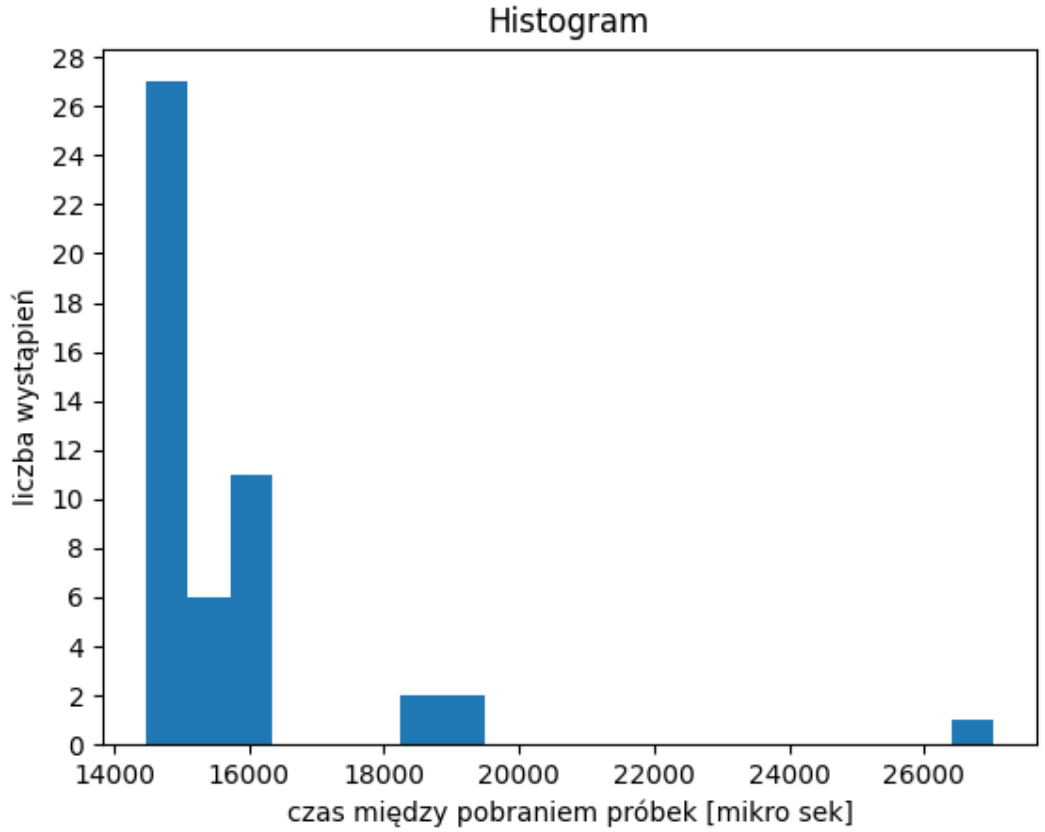


WNIOSKI:

W tym przypadku możemy zobaczyć, że tym razem klient 1 nie oczekuje dłużej niż klient 0. Czasy te się mniej więcej zrównały, co jest rezultatem wprowadzenia aktywnego oczekiwania dla obu klientów.

6. Właściwy pomiar czasu

Analiza danych zebranych w pliku *“server.txt”* dla wariantu:
aktywne oczekiwanie, 3 klientów, 2 rdzenie, bez obciążenia



WNIOSKI:

Widzimy, że czasy między pobraniem próbek mocno odbiegają od pożądanej wartości (10000). Dzieje się tak głównie dlatego, że pomiędzy pobraniem próbki, a jej wysłaniem występuje pewne opóźnienie, wynikające m.in. z faktu, że funkcja usleep, usypia proces na określony czas, ale zwykle mija jeszcze chwila, zanim proces odzyska czas procesora.

Aby wyeliminować to zjawisko, można by np. mierzyć czas od momentu pobrania próbki do jej wysłania i usypiać proces na czas pomniejszony o te opóźnienie.