

Raport z laboratorium 4 - 24.04.2024

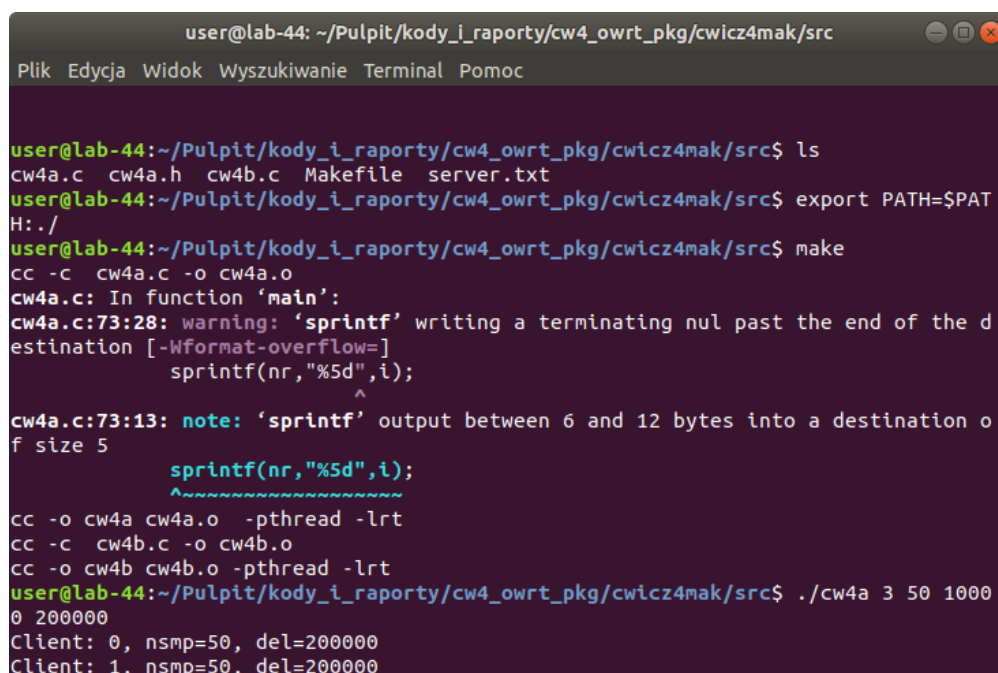
Sebastian Abramowski, 325142

Bogumił Stoma, 325233

Raport wygenerowany automatycznie na podstawie pliku raport.md

Przetestowanie działania programów na "gospodarzu"

Zaktualizowaliśmy zmienną środowiskową PATH, zbudowaliśmy programy przez **make**, wykonaliśmy program **cw4a**, aby upewnić się, że działa



```
user@lab-44: ~/Pulpit/kody_i_raporty/cw4_owrt_pkg/cwicz4mak/src
Plik Edycja Widok Wyszukiwanie Terminal Pomoc

user@lab-44:~/Pulpit/kody_i_raporty/cw4_owrt_pkg/cwicz4mak/src$ ls
cw4a.c cw4a.h cw4b.c Makefile server.txt
user@lab-44:~/Pulpit/kody_i_raporty/cw4_owrt_pkg/cwicz4mak/src$ export PATH=$PATH:./
user@lab-44:~/Pulpit/kody_i_raporty/cw4_owrt_pkg/cwicz4mak/src$ make
cc -c cw4a.c -o cw4a.o
cw4a.c: In function 'main':
cw4a.c:73:28: warning: 'sprintf' writing a terminating nul past the end of the destination [-Wformat-overflow=]
    sprintf(nr,"%5d",i);
                           ^
cw4a.c:73:13: note: 'sprintf' output between 6 and 12 bytes into a destination of size 5
    sprintf(nr,"%5d",i);
    ^~~~~~
cc -o cw4a cw4a.o -pthread -lrt
cc -c cw4b.c -o cw4b.o
cc -o cw4b cw4b.o -pthread -lrt
user@lab-44:~/Pulpit/kody_i_raporty/cw4_owrt_pkg/cwicz4mak/src$ ./cw4a 3 50 1000
0 200000
Client: 0, nsmp=50, del=200000
Client: 1, nsmp=50, del=200000
```

Zbudowanie pakietu dla OpenWRT

Skompilowaliśmy pakiet przy pomocy SDK posługując się naszym raportem z poprzedniego laboratorium i przetrzuciliśmy skompilowaną paczkę na RPi

Następnie zainstalowaliśmy nasz pakiet i uruchomiliśmy na RPi

```
user@lab-44: ~  
Plik Edycja Widok Wyszukiwanie Terminal Pomoc  
dev var  
etc worms_1.0-1_aarch64_cortex-a72.ipk  
lib www  
lib64 zad1.py  
lost+found zad2.py  
mnt zad22.py  
overlay zad3.py  
proc zad4.py  
rom zad5.py  
root@OpenWrt:/# opkg install cwicz4mak_1_aarch64_cortex-a72.ipk  
Installing cwicz4mak (1) to root...  
Configuring cwicz4mak.  
root@OpenWrt:/# ./cw4a 3 50 10000 200000  
/bin/ash: ./cw4a: not found  
root@OpenWrt:/# ./cw4a 3 50 10000 200000  
Client: 0, nsmp=50, del=200000  
Client: 1, nsmp=50, del=200000  
Client: 2, nsmp=50, del=200000  
Sample 0, client 0, delivery time: 27  
Sample 0, client 1, delivery time: 55Sample 0, client 2, delivery time: 67  
Sample 1, client 0, delivery time: 16Sample 1, client 2, delivery time: 20Sample 1, client 1, delivery time: 23
```

Przerzucanie plików

Przerzucanie plików pomiędzy hostem a RPi i odwrotnie było przeprowadzane przez serwer http używając polecenia `wget`

Ustalenie granicznej wartości czasu przetwarzania

Naszym zadaniem w tym podpunkcie było zaobserwowanie, dla jakich wartości czasu przetwarzania danych (liczba pętli symulująca przetworzenie danej wartości w programie `cw4b`), system przestaje nadążać przetwarzać dane w czasie rzeczywistym

Analizowaliśmy pliki `cli_*.txt`, aby zaobserwować dla jakich wartości czasu przetwarzania, opóźnienia widoczne będą rosły i będą wynosiły sporo (w porównaniu do poprzednich wyników)

Standardowe wykonanie programu `cw4a`, z takimi argumentami testowaliśmy różne warianty:

```
cw4a 3 200 10000 <CZAS_PRZETWARZANIA, zaczynaliśmy od 100 000>
```

Liczba rdzeni była zmieniana poprzez zmianę pliku `cmdline.txt` (`maxcpus`) oraz reboot systemu

Pełne obciążenie powodowaliśmy komendą

```
stress-ng --matrix 0 -t 5m
```

Sprawdzenie obciążenia rdzeni sprawdzaliśmy poprzez program `htop`

Wariant 1: 3 klientów, 1 rdzeń, pełne obciążenie

```
cw4a 3 200 10000 275000  
# powyżej 200 000 są widoczne wzrosty opóźnień, a później jest ich coraz więcej
```

Sprawdzenie obciążenia

```
user@lab-44: ~
Plik Edycja Widok Wyszukiwanie Terminal Pomoc

0[|||||100.0%] Tasks: 18, 0 thr, 47 kthr; 2 running
1[          ] Load average: 0.25 0.14 0.05
2[          ] Uptime: 00:03:49
3[          ]
Mem[||||] 20.2M/3.69G
Swp[      ] 0K/0K

Main I/O
PID USER PRI NI VIRT RES SHR S CPU%-MEM% TIME+ Command
2283 root 20 0 15412 1200 1004 R 397.4 0.0 0:07.17 stress-ng-matri
2284 root 20 0 1716 1476 1104 R 2.6 0.0 0:00.03 htop
1 root 20 0 1608 1020 828 S 0.0 0.0 0:00.28 /sbin/procd
120 ubus 20 0 1196 712 664 S 0.0 0.0 0:00.03 /sbin/ubusd
121 root 20 0 1228 1020 960 S 0.0 0.0 0:00.02 /bin/ash --logi
122 root 20 0 816 468 432 S 0.0 0.0 0:00.00 /sbin/askfirst
299 logd 20 0 1180 576 528 S 0.0 0.0 0:00.01 /sbin/logd -S 6
351 root 20 0 2124 1204 1012 S 0.0 0.0 0:00.00 /sbin/rpcd -s /
490 dnsmasq 20 0 1356 976 872 S 0.0 0.0 0:00.03 /usr/sbin/dnsma
552 root 20 0 1012 544 504 S 0.0 0.0 0:00.00 /usr/sbin/dropb
650 root 20 0 4316 736 680 S 0.0 0.0 0:00.00 /usr/sbin/hosta
651 root 20 0 4316 724 668 S 0.0 0.0 0:00.00 /usr/sbin/wpa_s
711 root 20 0 1796 1196 988 S 0.0 0.0 0:00.03 /sbin/netifd
root@OpenWrt:/#
```

Wariant 2: 3 klientów, 2 rdzenie, pełne obciążenie

```

cw4a 3 200 10000 450000
# 400 000 widać trochę, 450 000 widać duże opóźnienia

```

Sprawdzenie obciążenia

```
user@lab-44: ~
Plik Edycja Widok Wyszukiwanie Terminal Pomoc

0[|||||100.0%] Tasks: 19, 0 thr, 53 kthr; 4 running
1[|||||100.0%] Load average: 0.16 0.03 0.01
2[          ] Uptime: 00:00:46
3[          ]
Mem[||||] 22.7M/3.69G
Swp[      ] 0K/0K

Main I/O
PID USER PRI NI VIRT RES SHR S CPU%-MEM% TIME+ Command
1565 root 20 0 15416 1148 952 R 199.3 0.0 0:04.14 stress-ng-matri
1564 root 20 0 15416 1148 952 R 196.7 0.0 0:04.09 stress-ng-matri
1566 root 20 0 1724 1480 1104 R 2.6 0.0 0:00.05 htop
1 root 20 0 1608 1016 828 S 0.0 0.0 0:00.27 /sbin/procd
126 ubus 20 0 1196 716 664 S 0.0 0.0 0:00.02 /sbin/ubusd
127 root 20 0 1228 1020 960 S 0.0 0.0 0:00.01 /bin/ash --logi
128 root 20 0 816 532 496 S 0.0 0.0 0:00.00 /sbin/askfirst
307 logd 20 0 1180 564 520 S 0.0 0.0 0:00.00 /sbin/logd -S 6
359 root 20 0 2124 1208 1012 S 0.0 0.0 0:00.01 /sbin/rpcd -s /
498 dnsmasq 20 0 1356 976 872 S 0.0 0.0 0:00.00 /usr/sbin/dnsma
560 root 20 0 1012 544 504 S 0.0 0.0 0:00.00 /usr/sbin/dropb
658 root 20 0 4316 740 684 S 0.0 0.0 0:00.00 /usr/sbin/hosta
659 root 20 0 4316 740 684 S 0.0 0.0 0:00.00 /usr/sbin/wpa_s
F1help F2Setup F3Search F4filter F5Tree F6SortBy F7Ntice F8Ntice F9kill F10quit

```

Wariant 3: 3 klientów, 2 rdzenie, bez obciążenia

```

cw4a 3 200 10000 550000
# przy 500 000 widać trochę opóźnień, 550 000 się zaczynają stałe opóznianie

```

Wariant 4L 1 klient, 4 rdzenie, bez obciążenia

```

cw4a 3 200 10000 810000
# przy 800 000 się zaczęły większe opóźnia, dla 810 000 te opóźnienia były dużo
większe

```

Podsumowanie zadania nr. 3

Numer wariantu	Graniczny czas przetwarzania
1	275 000
2	450 000
3	550 000
4	810 000

Czy wyniki mają sens?

Wariant 3 działa bez obciążenia, więc jego wartość graniczna czasu przetwarzania powinna być większa od wariantu 2, co się zgadza.

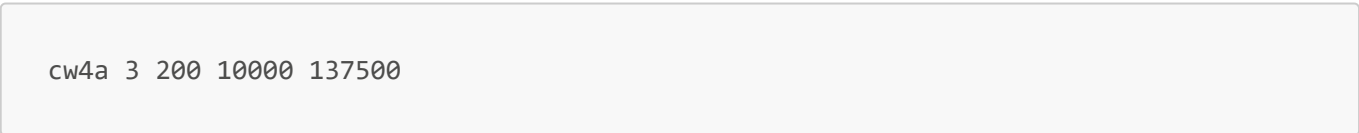
Wariant 1 działa na jednym obciążonym rdzeniu, a wariant 2 na dwóch obciążonych rdzeniach, więc wariant 2 powinien mieć wartość graniczną większą (przetwarzanie danych może trwać dłużej bez drastycznego wzrostu opóźnień) niż wariant 1, co też się zgadza.

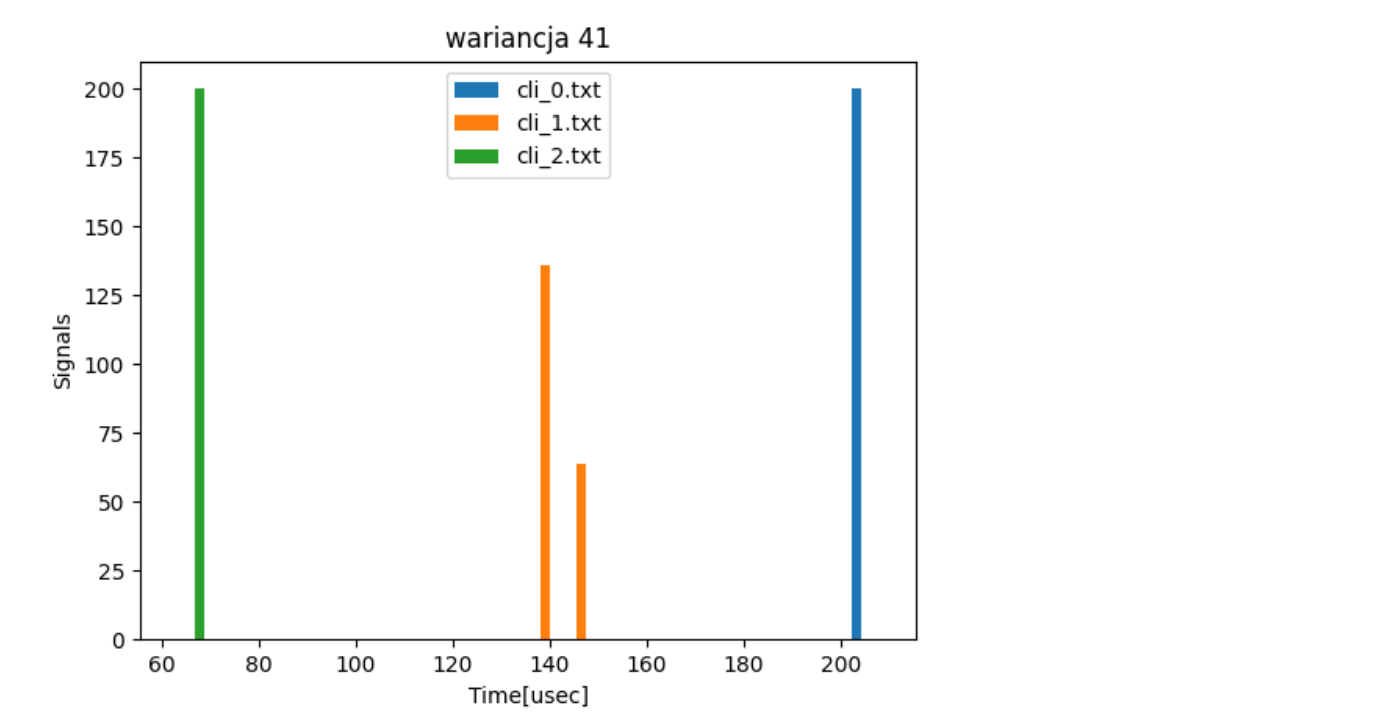
Wariant 4 ma najdłuższy graniczny czas przetwarzania, pewnie wynika to z faktu, że jeden klient nie ma konkurencji dostępu do zasobów dzielonych (bufora) w porównaniu do przypadku gdy mamy 3 klientów.

Rozkład czasu dostarczenia danych

Czas przetwarzania w tym podpunkcie wynosi połowe tych czasów co nam wyszły w podpunkcie powyżej. Przedstawione histogramy będą przedstawiały zależność opóźnienia dostarczenia danych [us] od liczby wystąpień

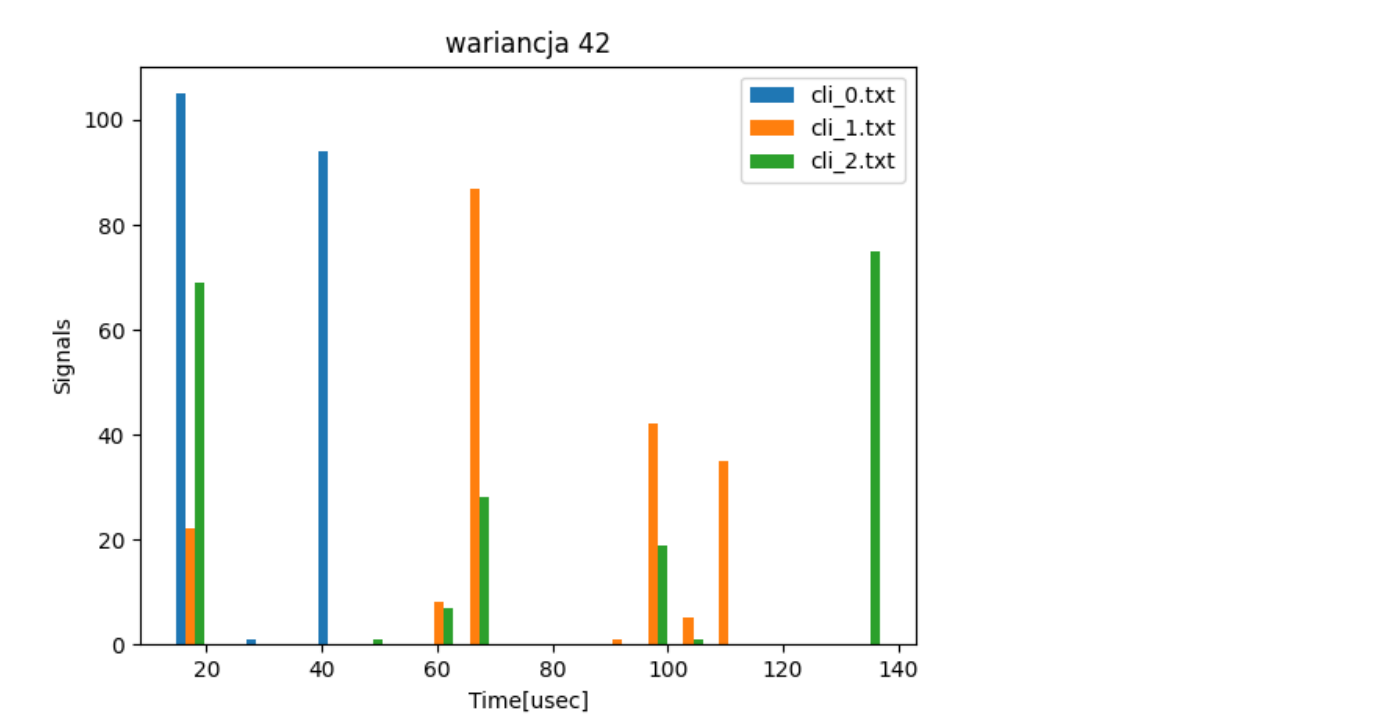
Wariant 1





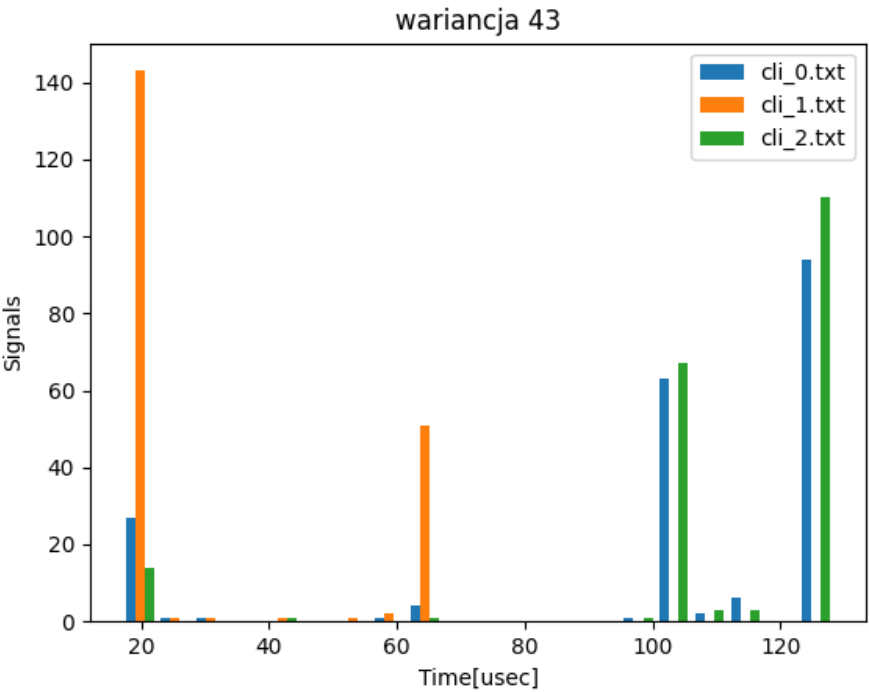
Wariant 2

```
cw4a 3 200 10000 225000
```



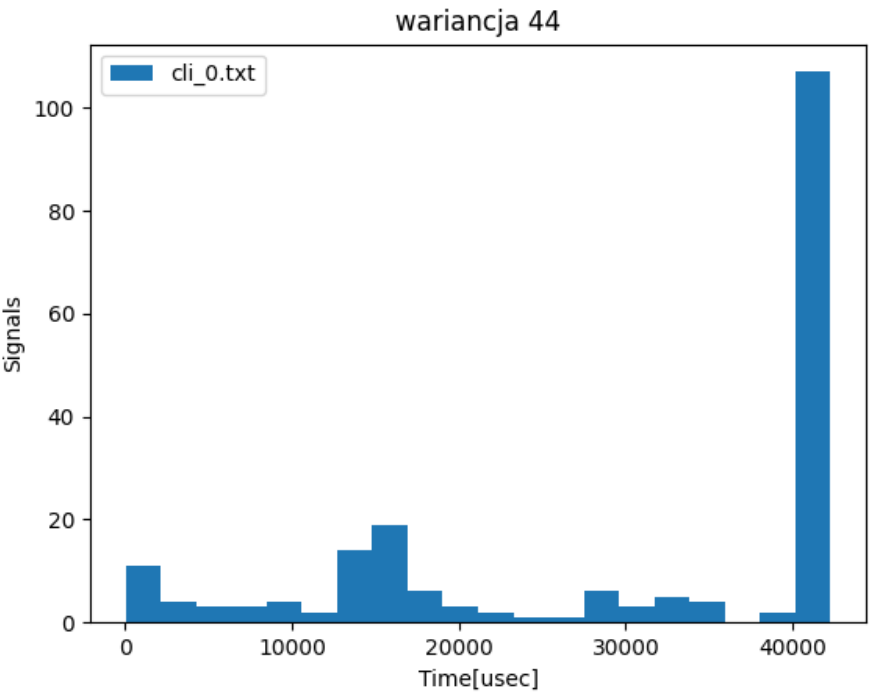
Wariant 3

```
cw4a 3 200 10000 275000
```



Wariant 4

```
cw4a 3 200 10000 405000
```



Aktywne oczekiwanie

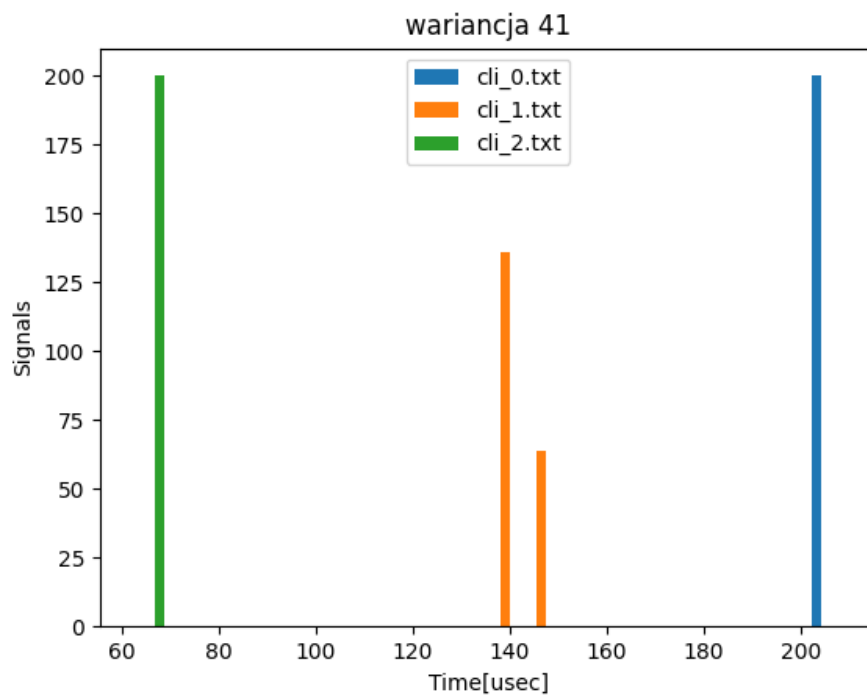
Ustawiliśmy aby można było ustawić tryb programu `cw4b` przez ustawienie odpowieniej zmiennej środowiskowej w terminalu przed uruchomieniem programu (nie chcieliśmy zmieniać argumentów, które program przyjmuje), zmienne środowiskowe:

- `ACTIVE_WAIT` - wszyscy kliencie będą oczekiwać aktywnie
- `ACTIVE_WAIT_CLIENT_NR_0` - klient nr. 0 będzie aktywnie oczekiwał

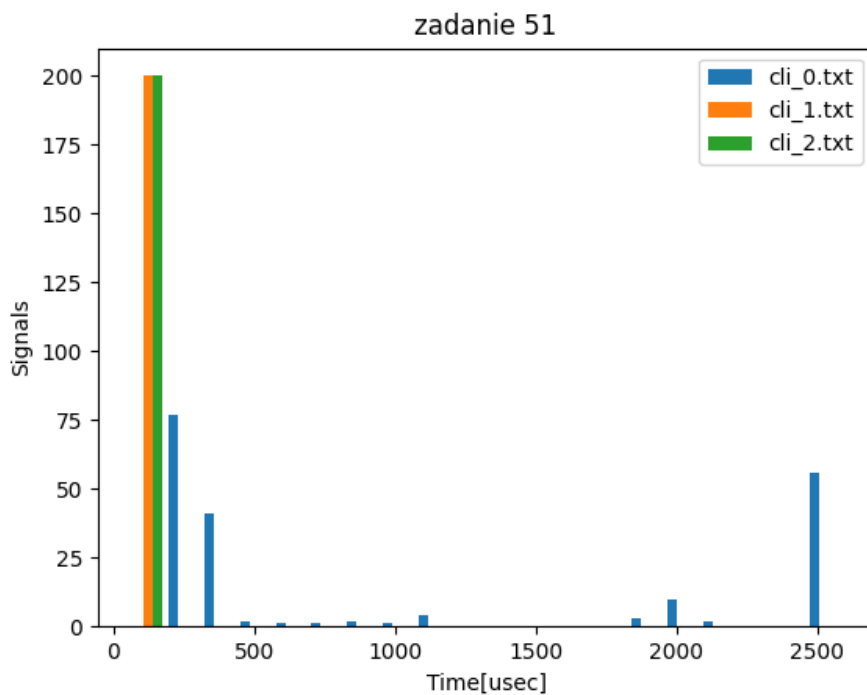
Jest to zrealizowane tak, że jeśli klient ma aktywnie oczekiwać, to nie zawiesza się na zmiennej warunkowej (`pthread_cond_wait(&rbuf->cvar,&rbuf->cvar_lock)`), ale kontynuuje swoje działanie, ponownie sprawdzając w pętli czy pojawiły się nowe dane

Jak aktywne oczekiwanie wpłynęło na opóźnienie dostarczenia danych?

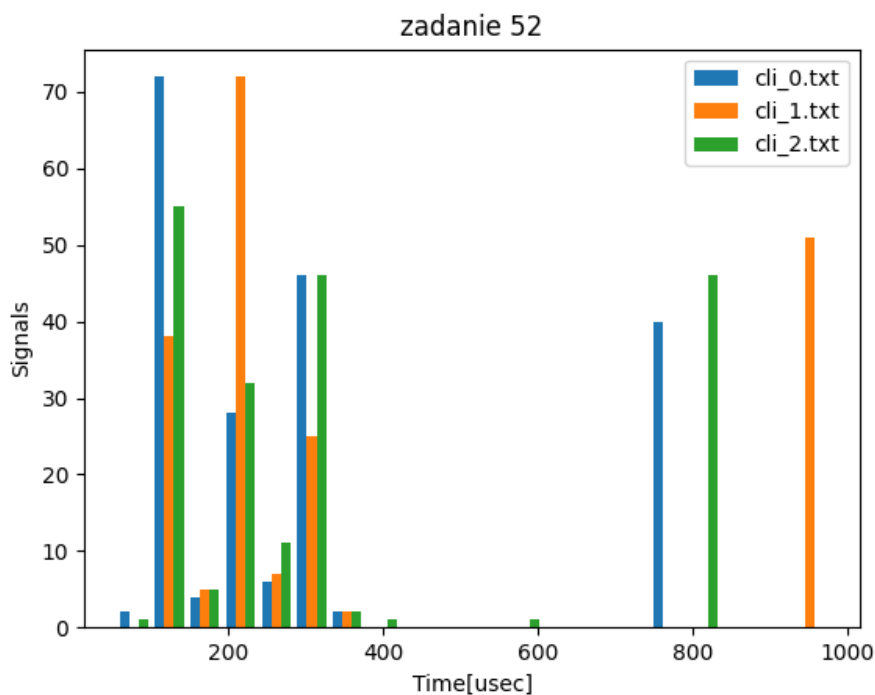
Badanie przeprowadziliśmy dla wariantu 1 z zadania nr. 4 (tego powyżej) - 3 klientów, 1 rdzeń, pełne obciążenie. Przypomnienie histogramu bez aktywnego oczekiwania:



Aktywne oczekiwanie klient nr. 0



Aktywne oczekiwanie wszystkich klientów



Obserwacje po zakończeniu zadania

Możemy zaobserwować, że czasy opóźnień dostarczania danych bez aktywnego oczekiwania były znacząco mniejsze. Na dodatek, wtedy mogliśmy zaobserwować w regularność i przewidywalność w czasach opóźnień.

Dla aktywnego oczekiwania klienta nr. 0, możemy zaobserwować, że opóźnienia tego klienta są znacznie większe (na oko w 1/3 przypadków) niż bez. Na dodatek, klienci nr. 1 i nr. 2 mają zwiększone czasy opóźnień (pewnie wynika to z faktu, że proces klienta nr. 0 bez sensu wykorzystuje czas procesora).

W aktywnym oczekiwaniu wszystkich klientów także możemy zobaczyć powyższy obserwację. Mniejsza regularność, większe czasy opóźnień dla wszystkich klientów. Co ciekawe, w mniej więcej dwóch piątych przypadków, klient nr. 0 ma mniejsze opóźnienie przetwarzania niż w przypadku bez aktywnego oczekiwania (chodzi o niebieski pasek na wykresie z tytułem **zadanie 52** dla wartości czasu mniej więcej 90 us)

Właściwy pomiar czasu

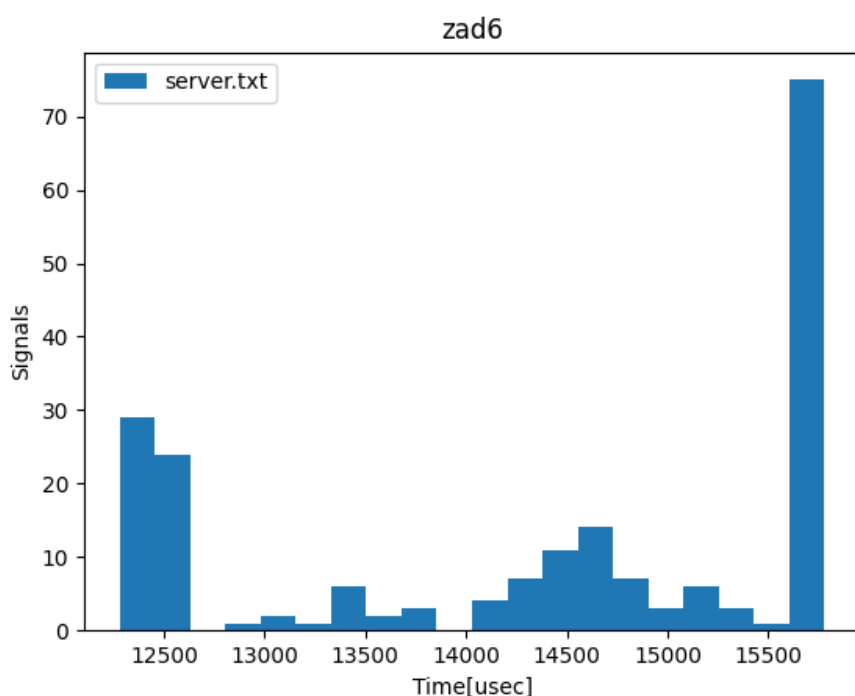
Badania w tym zadaniu przeprowadziliśmy dla wariantu 3 z zadania nr. 3 (**cw4a 3 200 10000 550000**)

Problem w programie **cw4a** polegał na tym, że w każdym wykonaniu pętli, było spanie w programie o okres równy czasu próbkowania. Czyli np. jeśli byśmy mieli okres próbkowania 1s, to czas pomiędzy wygenerowaniem kolejnych dwóch próbek wynosiłby 1s + czas przetwarzania próbki danych. Trzeba było zastosować korektę okresu próbkowania w pętli, abyśmy usypiali na tyle ile to jest konieczne.

Czyli np. jeśli okres próbkowania wynosi 1s oraz przetwarzamy przez 0.5s daną próbkę danych (u nas jest to aktualny czas), to w następnym kroku pętli, będziemy spali 0.5s zamiast 1s. W ten sposób uzyskamy program, w którym okres między pobraniami zestawów próbek jest mniej więcej właściwy.

Wariant 3 z zadania nr. 3 przed poprawieniem programu

Wykresy przedstawiają czas pomiędzy pobraniami zestawów próbek w mikrosekundach oraz ile razy te czasy wystąpiły.



Wariant 3 z zadania nr. 3 przed poprawieniem programu

W tym przypadku widzimy, że czas jest dużo bliższy okresowi próbowania, który wnosił **10000** us w naszym przypadku.

