

[SCZR] Dokumentacja końcowa

Adrian Brodzik

Bartosz Świtalski

Piotr Frątczak

20 grudnia 2020

Opis zadania

Celem projektu jest stworzenie systemu czasu rzeczywistego przetwarzającego obrazy z kamierki wbudowanej lub USB.

Realizacja

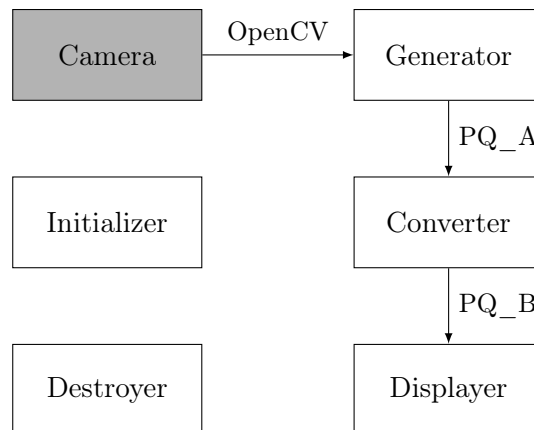
Stworzono system rejestrujący za pomocą kamierki pozycję kciuka obracanego przez użytkownika. Na podstawie pozycji palca na wyjściu w czasie rzeczywistym wytwarzana jest wartość, która jest tangensem kąta nachylenia kciuka do poziomu.

Specyfikacja wymagań

Implementowany system pracuje pod kontrolą systemu operacyjnego Ubuntu. Komunikacja międzyprocesowa odbywa się za pomocą kolejek priorytetowych. Inicjowane są odpowiednie części systemu (producenci, konsumenci) za pomocą mechanizmów realizujących dostęp do pamięci współdzielonej. Realizacja wykorzystuje standard `System V`, w tym biblioteki `sys/sem.h`, `sys/shm.h`.

W celu przeprowadzenia analizy wydajności i ewentualnych optymalizacji system będzie funkcjonował na określonej liczbie rdzeni. Do analizy wydajności wykorzystane zostaną także stemple czasowe, będące elementami przetwarzanych wiadomości (struktur).

Schemat systemu



Oznaczenia

Operacja **pop** pobiera obraz z kamery. Operacje **down** i **up** odpowiednio obniżają i podnoszą wartość semafora. Operacje **push** i **pop** odpowiednio dodają i usuwają element kolejki. Operacja **process** przetwarza obraz. Operacja **display** wyświetla wartość końcową.

Semafor **EMPTY** wskazuje wolne miejsce w kolejce. Semafor **FULL** wskazuje obecność co najmniej jednego elementu w kolejce. Semafor **MUTEX** wskazuje obecność procesu w sekcji krytycznej. Każda z kolejek **PQ** ma swoje semafory.

Initializer

Proces odpowiedzialny za inicjalizację i alokację pamięci współdzielonej, w tym semaforów i kolejek priorytetowych.

Generator

Proces odpowiedzialny za pobranie obrazów z kamery i wstępne przetwarzanie (prycinanie i skalowanie). Realizacja za pomocą biblioteki [OpenCV](#).

```
while(true):
    img := input()

    down(EMPTY_A)
    down(MUTEX_A)

    push(PQ_A, img)

    up(MUTEX_A)
    up(FULL_A)
```

Converter

Proces przetwarzający dane odebrane od procesu **Generator** i przekazujący je do procesu **Displayer**. **Converter** odbiera dane z kolejki **PQ_A**, wyznacza kąt nachylenia prostej przechodzącej przez dwa punkty i przekazuje je do kolejki **PQ_B**. Pierwszy punkt jest środkiem ciężkości maski dłoni, a drugi jest najbardziej oddalonym punktem otoczki wypukłej dłoni. W ten sposób znane jest nachylenie kciuka lub innego palca sterującego.

```
while(true):
    down(FULL_A)
    down(MUTEX_A)

    img := pop(PQ_A)

    up(MUTEX_A)
    up(EMPTY_A)

    val := process(img)

    down(EMPTY_B)
    down(MUTEX_B)

    push(PQ_B, val)

    up(MUTEX_B)
    up(FULL_B)
```

Displayer

Proces wyświetlający wartość końcową w oknie za pomocą biblioteki [gtkmm](#). Wartości odbierane są z kolejki **PQ_B**.

```
while(true):
    down(FULL_B)
    down(MUTEX_B)

    val := pop(PQ_B)

    up(MUTEX_B)
    up(EMPTY_B)

    display(val)
```

Destroyer

Proces odpowiedzialny za dealokację pamięci współdzielonej, w tym semaforów i kolejek priorytetowych.

Analiza wydajności

Metoda

`Displayer` odbiera wiadomości z kolejki `PQ_B`, które zawierają stemple czasowe: utworzenia, wysłania i odebrania wiadomości A oraz B (wiadomość A przekazuje informacje między pierwszymi dwoma procesami, następnie przetworzone informacje z wiadomości A przekazywane są do wiadomości B, która zbiera stemple czasowe wiadomości A oraz swoje). Stemple są zapisywane do pliku po przekazaniu wiadomości B do ostatniego procesu.

Po ukończeniu wykonywania programu zebrane dane zostały poddane analizie przez zewnętrzny skrypt, który był odpowiedzialny za stworzenie plików przedstawiające odpowiednie wyniki w formie graficznej.

Zmierzono: całkowity czas między utworzeniem wiadomości i wyświetleniem danych z wiadomości przez `Displayer`, czas spędzony w kolejce, czas spędzony w każdym z procesów.

Obliczono średnie i odchylenia standardowe zmierzonych czasów w zależności od ustawionych parametrów systemu.

Obciążenie systemu

System czasu rzeczywistego działa pod kontrolą systemu operacyjnego Ubuntu, stąd konieczność implementacji dodatkowego obciążenia, w celu zbadania poprawności działania systemu czasu rzeczywistego. Do obciążenia systemu użyto programu `stress-ng`. Za jego pomocą jednocześnie CPU zostało obciążone zestawem wielu operacji, wywołanych było także wiele funkcji `mmap/munmap` zapisując do alokowanej pamięci oraz było tworzonych wiele timerów na raz z częstotliwością 1Mhz.

Manipulacja priorytetami procesów

W celu manipulacji priorytetami procesów zaprojektowanego systemu wykorzystano program `chrt`. Procesom systemu czasu rzeczywistego nadano politykę szeregowania `SCHED_FIFO`.

Manipulacja rozmiarami kolejek

Zbadano wpływ rozmiaru kolejek na wydajność reszty systemu. Przetestowane rozmiary kolejek to: 1, 5, 10, 20, 50.

Spostrzeżenia

Konwolucyjna sieć neuronowa wykonana w `PyTorch` odpowiedzialna za przetwarzanie obrazów w procesie `Converter` miała bardzo negatywny wpływ na wydajność całego systemu rzeczywistego, gdyż wprowadzała opóźnienie rzędu 2 – 3 sekund.

Ze względu na ograniczenia czasowe systemu realizacja przetwarzania obrazów została wykonana wyłącznie w `OpenCV`.

Testowanie

Testowano cztery scenariusze:

- procesor nieobciążony, priorytet normalny systemu RT
- procesor obciążony, priorytet normalny systemu RT
- procesor obciążony, priorytet wysoki systemu RT
- procesor obciążony, priorytet wysoki najwolniejszego procesu w systemie RT (converter)

Wyniki testów zostały omówione w dalszej części tego sprawozdania.

Wyniki

Objaśnienia

raw idle - czas oczekiwania wiadomości (danych pobranych z kamery) na push do PQ_A

raw queue - czas pobytu wiadomości w kolejce PQ_A

raw→proc - czas przetwarzania wiadomości na dane wyjściowe

proc idle - czas oczekiwania wiadomości (danych wyjściowych) na push do PQ_A

proc queue - czas pobytu wiadomości w kolejce PQ_B

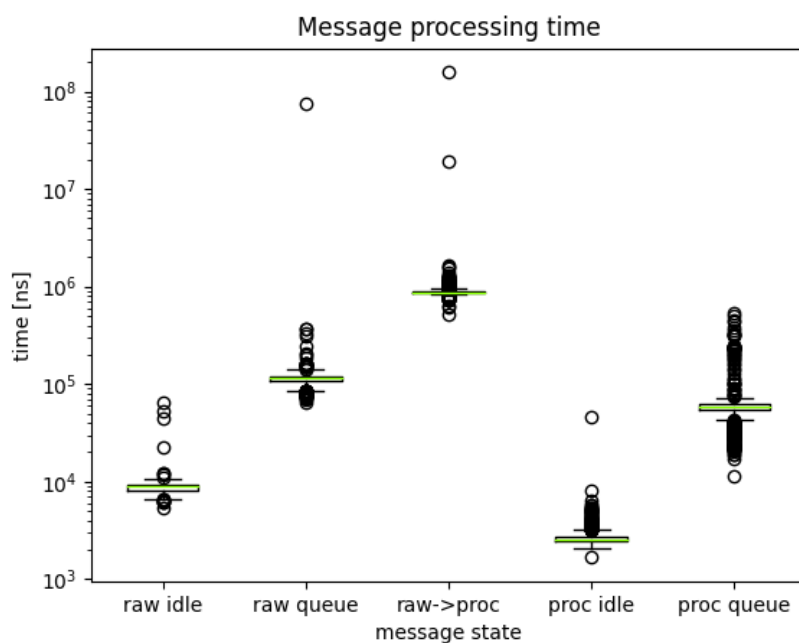
Jednostki

Wyniki zostały podane w nanosekundach. Wyniki w tabelach przedstawiono w notacji wykładniczej, lecz wykładniki zostały przekształcone do wielokrotności liczby 3, w celu podkreślenia jak zmieniają się jednostki czasu, na których operowano (ns, μ s, ms, s).

System nieobciążony, priorytet normalny

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$8.88 \cdot 10^3$	$0.18 \cdot 10^6$	$1.04 \cdot 10^6$	$2.7 \cdot 10^3$	$0.06 \cdot 10^6$
odchylenie standardowe	$2.47 \cdot 10^3$	$2.17 \cdot 10^6$	$4.53 \cdot 10^6$	$1.34 \cdot 10^3$	$0.04 \cdot 10^6$

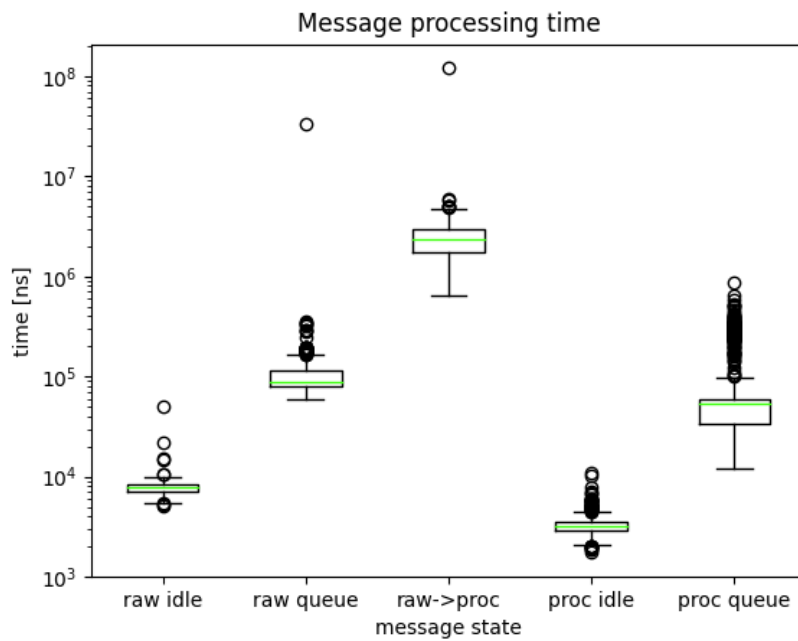
Tablica 1: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 1: Rozmiar kolejki = 5, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$7.83 \cdot 10^3$	$0.13 \cdot 10^6$	$2.49 \cdot 10^6$	$3.3 \cdot 10^3$	$0.08 \cdot 10^6$
odchylenie standardowe	$1.53 \cdot 10^3$	$0.97 \cdot 10^6$	$3.48 \cdot 10^6$	$0.68 \cdot 10^3$	$0.09 \cdot 10^6$

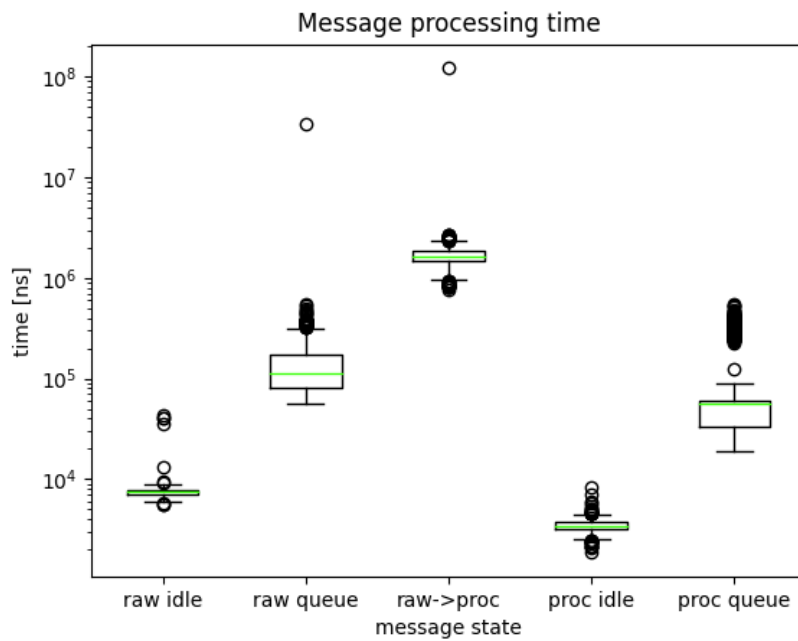
Tablica 2: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 2: Rozmiar kolejki = 10, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$7.59 \cdot 10^3$	$0.17 \cdot 10^6$	$1.77 \cdot 10^6$	$3.49 \cdot 10^3$	$0.08 \cdot 10^6$
odchylenie standardowe	$1.02 \cdot 10^3$	$1.25 \cdot 10^6$	$3.63 \cdot 10^6$	$2.11 \cdot 10^3$	$0.13 \cdot 10^6$

Tablica 3: Czas przetwarzania wiadomości w nanosekundach [ns].

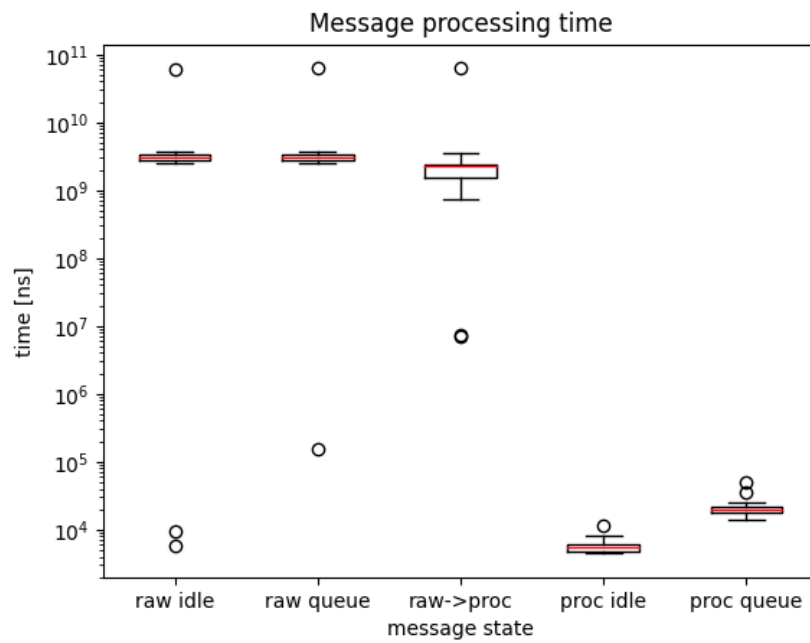


Rysunek 3: Rozmiar kolejki = 50, czas pomiaru 120 sekund.

System obciążony, priorytet normalny

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$6.01 \cdot 10^9$	$6.18 \cdot 10^9$	$5.23 \cdot 10^9$	$5.90 \cdot 10^3$	$0.02 \cdot 10^6$
odchylenie standardowe	$0.01 \cdot 10^{12}$	$0.01 \cdot 10^{12}$	$0.01 \cdot 10^{12}$	$1.71 \cdot 10^3$	$7.87 \cdot 10^3$

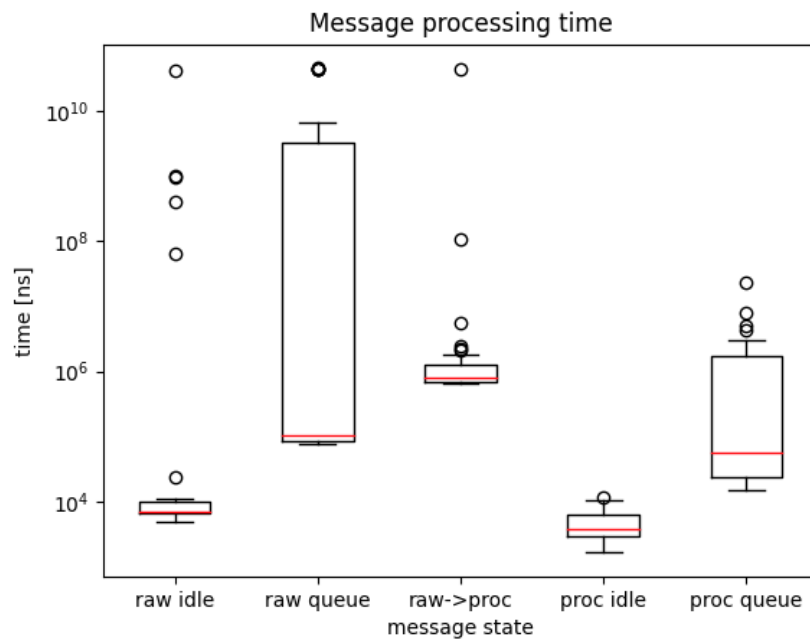
Tablica 4: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 4: Rozmiar kolejki = 1, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$1.17 \cdot 10^9$	$6.83 \cdot 10^9$	$1.20 \cdot 10^9$	$4.71 \cdot 10^3$	$1.70 \cdot 10^6$
odchylenie standardowe	$6.71 \cdot 10^9$	$0.02 \cdot 10^{12}$	$7.29 \cdot 10^9$	$2.58 \cdot 10^3$	$3.91 \cdot 10^6$

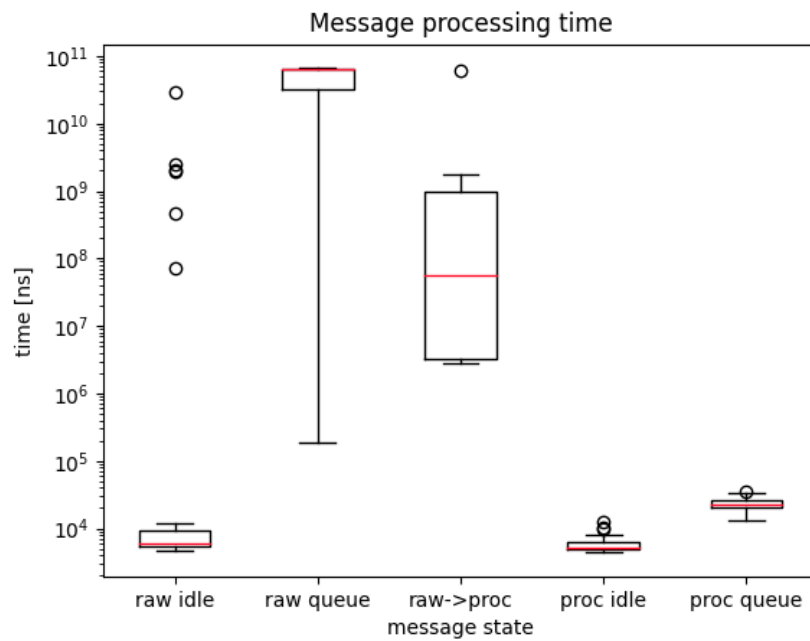
Tablica 5: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 5: Rozmiar kolejki = 5, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$1.11 \cdot 10^9$	$0.05 \cdot 10^3$	$2.31 \cdot 10^9$	$5.96 \cdot 10^3$	$0.02 \cdot 10^6$
odchylenie standardowe	$4.97 \cdot 10^9$	$0.02 \cdot 10^3$	$0.01 \cdot 10^3$	$1.83 \cdot 10^3$	$5.68 \cdot 10^3$

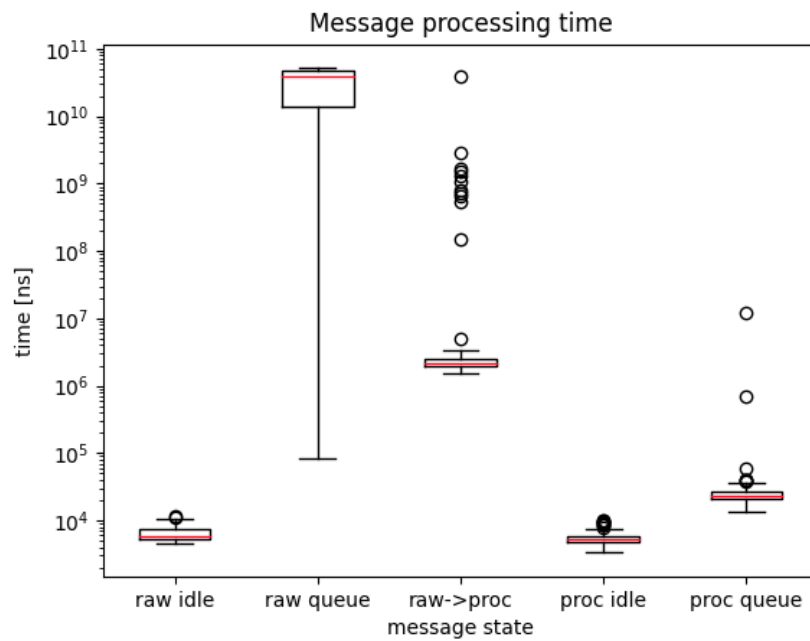
Tablica 6: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 6: Rozmiar kolejki = 20, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$6.56 \cdot 10^3$	$0.03 \cdot 10^{12}$	$0.80 \cdot 10^9$	$5.75 \cdot 10^3$	$0.23 \cdot 10^6$
odchylenie standardowe	$1.80 \cdot 10^3$	$0.01 \cdot 10^{12}$	$4.85 \cdot 10^9$	$1.62 \cdot 10^3$	$1.50 \cdot 10^6$

Tablica 7: Czas przetwarzania wiadomości w nanosekundach [ns].

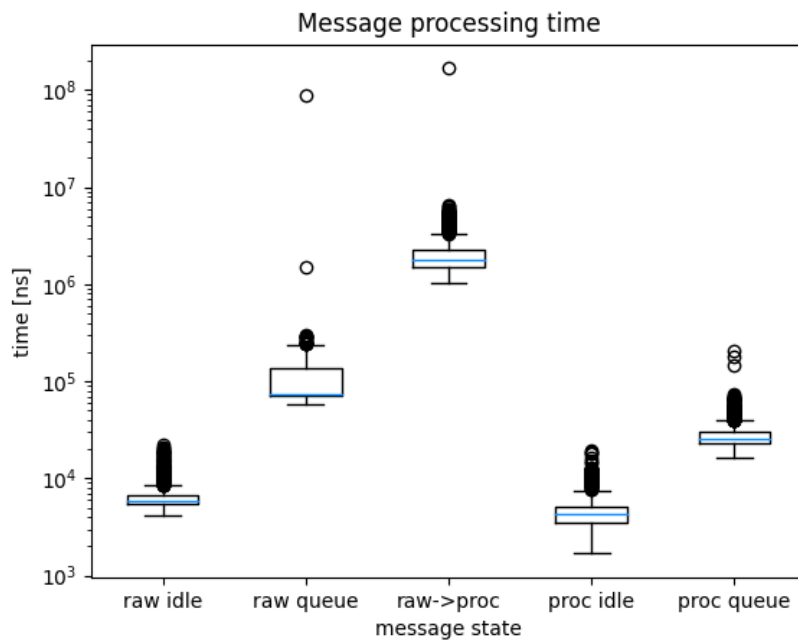


Rysunek 7: Rozmiar kolejki = 50, czas pomiaru 120 sekund.

System obciążony, priorytet wysoki

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$6.86 \cdot 10^3$	$0.18 \cdot 10^6$	$2.29 \cdot 10^6$	$4.72 \cdot 10^3$	$0.03 \cdot 10^6$
odchylenie standardowe	$2.64 \cdot 10^3$	$2.56 \cdot 10^6$	$4.85 \cdot 10^6$	$2.21 \cdot 10^3$	$0.01 \cdot 10^6$

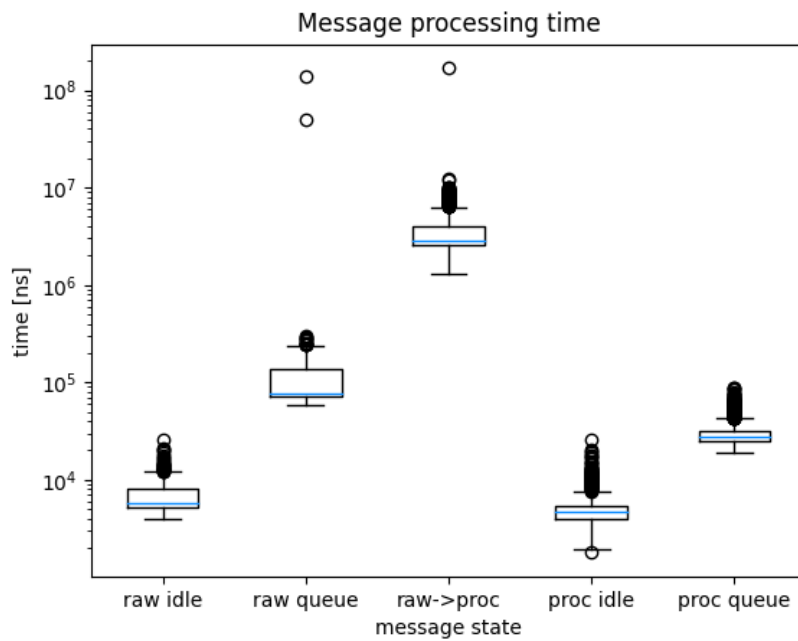
Tablica 8: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 8: Rozmiar kolejki = 20, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$6.60 \cdot 10^3$	$0.25 \cdot 10^6$	$3.73 \cdot 10^6$	$5.07 \cdot 10^3$	$0.03 \cdot 10^6$
odchylenie standardowe	$2.25 \cdot 10^3$	$4.18 \cdot 10^6$	$5.04 \cdot 10^6$	$2.27 \cdot 10^3$	$9.77 \cdot 10^3$

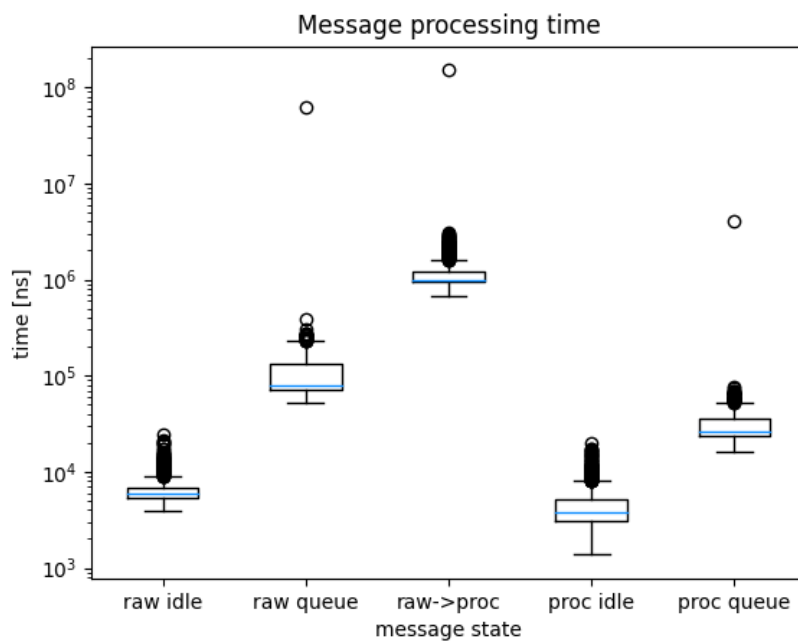
Tablica 9: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 9: Rozmiar kolejki = 10, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$6.83 \cdot 10^3$	$0.15 \cdot 10^6$	$1.35 \cdot 10^6$	$4.70 \cdot 10^3$	$0.03 \cdot 10^6$
odchylenie standardowe	$2.51 \cdot 10^3$	$1.76 \cdot 10^6$	$4.32 \cdot 10^6$	$2.58 \cdot 10^3$	$0.12 \cdot 10^6$

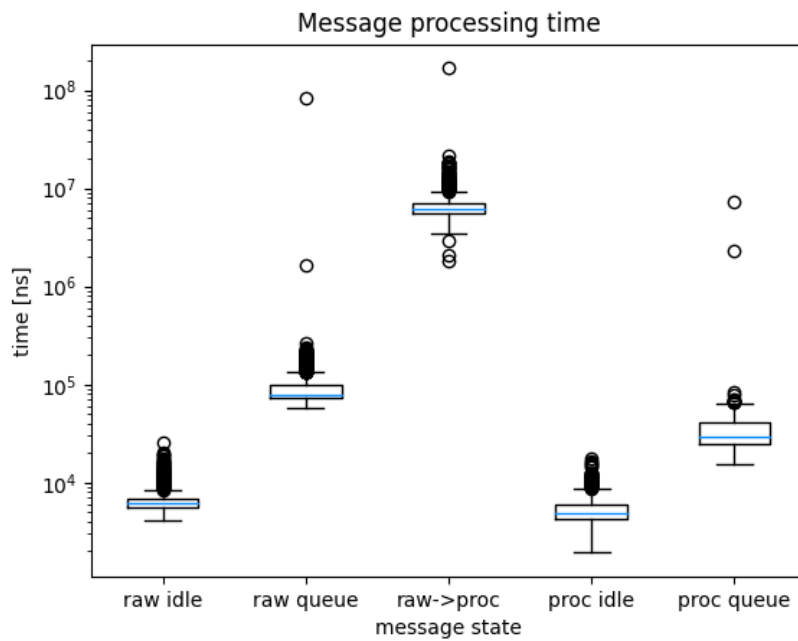
Tablica 10: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 10: Rozmiar kolejki = 5, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$6.74 \cdot 10^3$	$0.17 \cdot 10^6$	$6.90 \cdot 10^6$	$5.55 \cdot 10^3$	$0.04 \cdot 10^6$
odchylenie standardowe	$2.21 \cdot 10^3$	$2.39 \cdot 10^6$	$5.10 \cdot 10^6$	$1.93 \cdot 10^3$	$0.22 \cdot 10^6$

Tablica 11: Czas przetwarzania wiadomości w nanosekundach [ns].

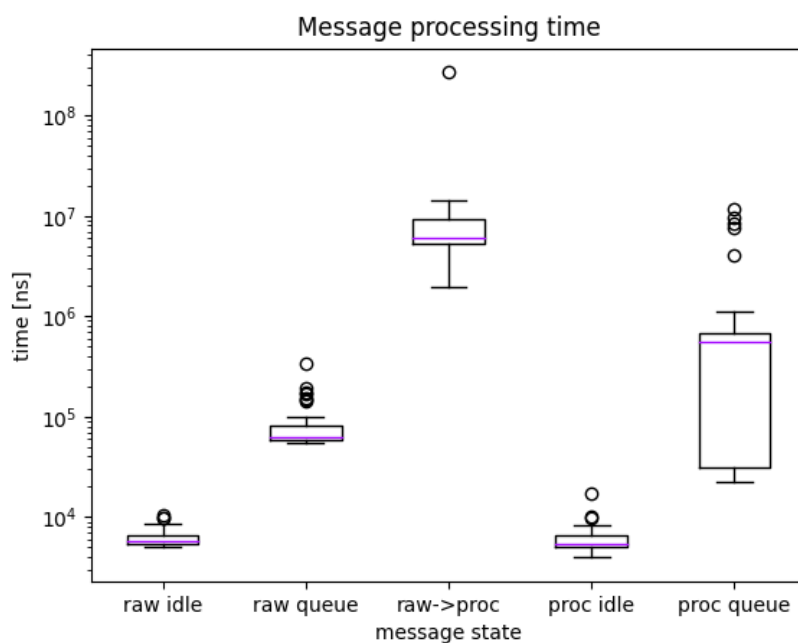


Rysunek 11: Rozmiar kolejki = 1, czas pomiaru 120 sekund.

System obciążony, priorytet wysoki converter

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$6.32 \cdot 10^3$	$0.09 \cdot 10^6$	$0.01 \cdot 10^9$	$6.21 \cdot 10^3$	$1.61 \cdot 10^6$
odchylenie standardowe	$1.43 \cdot 10^3$	$0.06 \cdot 10^6$	$0.04 \cdot 10^9$	$2.43 \cdot 10^3$	$3.01 \cdot 10^6$

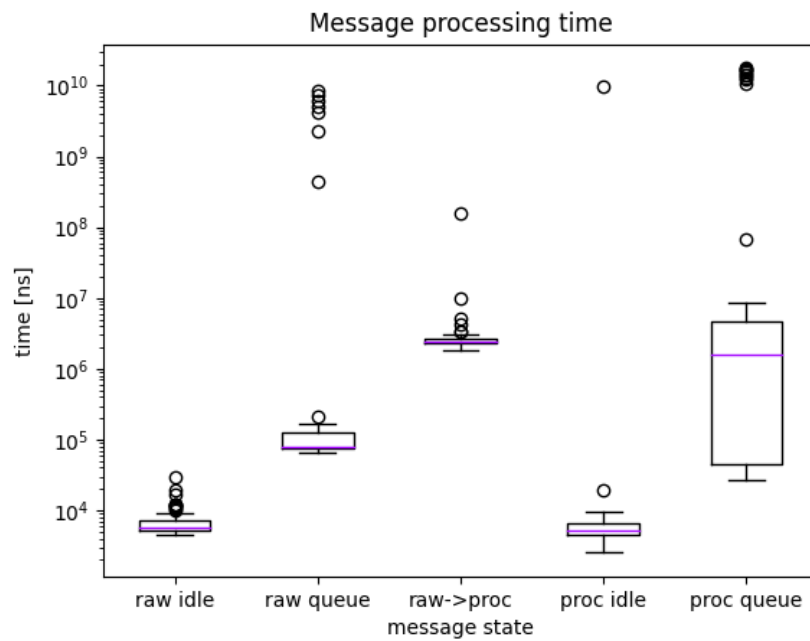
Tablica 12: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 12: Rozmiar kolejki = 50, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$7.32 \cdot 10^3$	$0.55 \cdot 10^9$	$5.19 \cdot 10^6$	$0.16 \cdot 10^9$	$2.42 \cdot 10^9$
odchylenie standardowe	$4.07 \cdot 10^3$	$1.77 \cdot 10^9$	$0.02 \cdot 10^9$	$1.26 \cdot 10^9$	$5.52 \cdot 10^9$

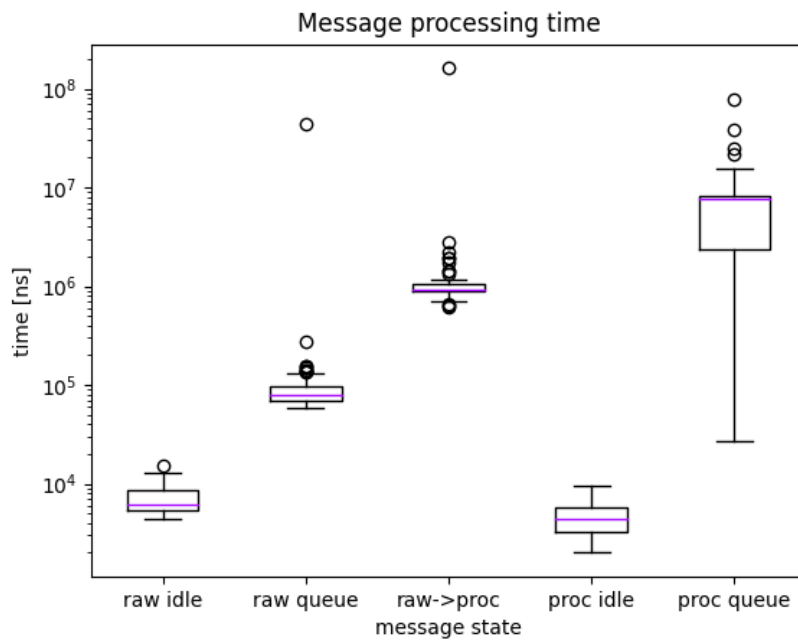
Tablica 13: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 13: Rozmiar kolejki = 10, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$6.96 \cdot 10^3$	$0.80 \cdot 10^6$	$3.67 \cdot 10^6$	$4.81 \cdot 10^3$	$8.06 \cdot 10^6$
odchylenie standardowe	$2.32 \cdot 10^3$	$5.49 \cdot 10^6$	$0.02 \cdot 10^9$	$1.99 \cdot 10^3$	$0.01 \cdot 10^9$

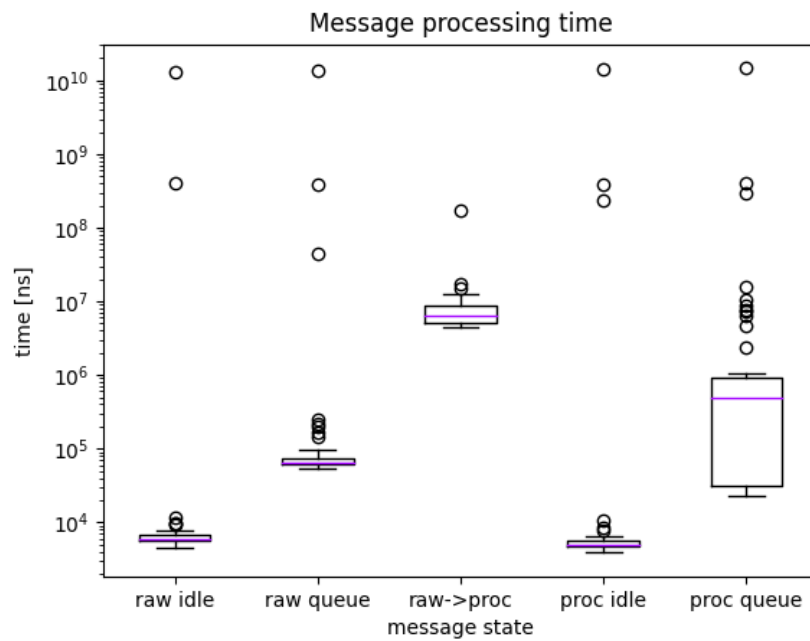
Tablica 14: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 14: Rozmiar kolejki = 5, czas pomiaru 120 sekund.

stan wiadomości	raw idle	raw queue	raw→proc	proc idle	proc queue
średni czas przetwarzania	$0.28 \cdot 10^9$	$0.29 \cdot 10^9$	$0.01 \cdot 10^9$	$0.31 \cdot 10^9$	$0.32 \cdot 10^9$
odchylenie standardowe	$1.86 \cdot 10^9$	$1.90 \cdot 10^9$	$0.02 \cdot 10^9$	$2.06 \cdot 10^9$	$2.09 \cdot 10^9$

Tablica 15: Czas przetwarzania wiadomości w nanosekundach [ns].



Rysunek 15: Rozmiar kolejki = 1, czas pomiaru 120 sekund.

Wnioski


Według zebranych wyników optymalizacja działania zaimplementowanego systemu RT miała duże znaczenie na szybkości odpowiedzi systemu na działanie użytkownika. W przypadku braku optymalizacji przy znacznym obciążeniu systemu operacyjnego Ubuntu czasy przetwarzania wiadomości znacznie się wydłużały, nawet o 1 do 10 sekund, podczas gdy bez obciążenia średni czas od pobrania danych z kamery do otrzymania wyniku ostatniego procesu nie przekraczał milisekundy. Najbardziej wymagającym czasowo procesem był `converter`, ponieważ wykonywał najwięcej operacji na danych.

Można również zauważyć wpływ czasu przetwarzania danych przez `converter` na długość czasu oczekiwania wiadomości w kolejce `PQ_A`. Zaszła widoczna różnica po obciążeniu systemu, ponieważ przy braku obciążenia czas oczekiwania w obu kolejkach był porównywalny, jednak w przypadku systemu obciążonego bez nadanego wyższego priorytetu czas oczekiwania w pierwszej kolejce zwiększył się o kilka rzędów i stał się podobny do czasu przetwarzania informacji w procesie `converter`. Stało się tak, gdyż `generator` bardzo szybko wstawiał nowe dane do kolejki, a proces `converter` pobierał z kolejki pojedyncze wiadomości i przetwarzał je o kilka rzędów wielkości dłużej, dlatego kolejka była cały czas zapełniona i czas spędzany przez wiadomości w kolejce wydłużył się wielokrotnie.

Po zmianie sposobu szeregowania dla całego systemu RT przy obciążonym systemie operacyjnym uzyskane dane pokrywały się z sytuacją, w której system operacyjny nie był obciążony, a nasz system RT pracował na normalnym priorytecie.

W sytuacji, kiedy system operacyjny był obciążony, a wysoki priorytet został ustalony jedynie dla procesu `converter`, czas reakcji systemu RT był opóźniany jedynie przez proces `displayer`, który w dalszym ciągu działał na priorytecie normalnym i stał się bardziej kosztowny czasowo w porównaniu do pozostałych procesów, zależących od pracy procesu `converter`.

Powiązane linki

 [Repozytorium projektowe](#)