Dokumentacja do programów symulacyjnych algorytmów planowania czasu procesora i zastępowania stron

Autor Valiantsin Lambin

Spis treści:

Algorytmy planowania czasu procesora

Opis algorytmów

FCFS lub First-Come, First-Served, to prosty algorytm planowania w dziedzinie systemów operacyjnych. Zasada FCFS polega na tym, że procesy są obsługiwane w kolejności, w jakiej wchodzą do systemu. Pierwsze weszło-pierwsze zostało obsłużone (First-Come, First-Served).

SJF lub Shortest job First to algorytm planowania w systemach operacyjnych, w którym procesy są wybierane do wykonania na podstawie ich zamierzonego czasu wykonywania. Procesy o najniższym oczekiwanym czasie wykonywania są wybierane jako pierwsze do obsługi. Algorytm zaimplementowany w dwóch wersjach: wywłaszczeniowej i nie wywłaszczeniowej. Różnica polega na tym, że **SJF Preemptive** (wywłaszczeniowy) może przerwać wykonywanie bieżącego procesu w dowolnym momencie, jeśli pojawi się proces o krótszym czasie wykonywania. **SJF Non-Preemptive** (niewywłaszczeniowy) algorytm planowania, wręcz przeciwnie, nie przerywa wykonywania bieżącego procesu przed jego zakończeniem. Dopiero po zakończeniu bieżącego procesu wybierany jest nowy proces do wykonania.

Opis generatora procesów

Moduł **proces_generator.py** zawiera implementację generatora procesów, mianowicie trzy funkcje służące do generowania procesów o różnych rozkładach czasów wykonania. Dla wygenerowania losowych wartości czasów nadejścia, wykonania wykorzystany moduł losowy w bibliotece **numpy**:

generate_processes_normal: Ta funkcja generuje procesy o czasach wykonania podążających za rozkładem normalnym. Czasy przyjścia są losowo wybierane z określonego zakresu, a czasy wykonania są pobierane z rozkładu normalnego o podanej średniej (**avg_execution_time**) i odchyleniu standardowym (**std_execution_time**). Otrzymane czasy wykonania są konwertowane na liczby całkowite. *Zastosowanie rozkładu normalnego jest irracjonalne. Zadanie wymaga, aby czas wykonania był liczbą całkowitą. Równomierny rozkład - rozkład ciągły. Próba konwersji wygenerowanych wartości float do int wpłynie*

na kształt rozkłady. Z tego powodu w eksperymentach stosuje się rozkład dwumianowy (rozkład Bernoulliego).

generate_processes: Ta funkcja generuje procesy o czasach wykonania podążających za rozkładem jednorodnym. Czasy przyjścia są losowo wybierane z określonego zakresu, a czasy wykonania są pobierane z rozkładu jednorodnego o podanym zakresie (**execution_time_range**).

generate_processes_binomial: Ta funkcja generuje procesy o czasach wykonania podążających za rozkładem dwumianowym. Czasy przyjścia są losowo wybierane z określonego zakresu, a czasy wykonania są pobierane z rozkładu dwumianowego o parametrach n (łączna liczba prób) i p (prawdopodobieństwo sukcesu). Średni czas wykonania jest używany do obliczenia p, a otrzymane czasy wykonania są liczbami całkowitymi.

Opis procedury testowania

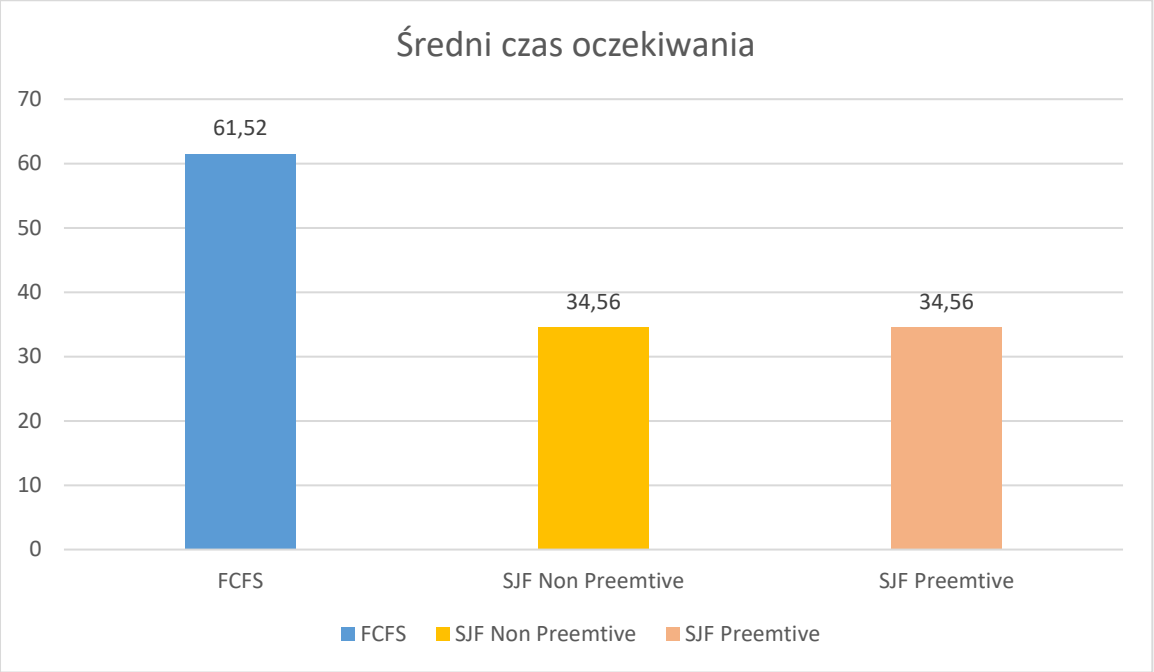
Implementację poniższych testów zawierają funkcje **test1()** i **test2()** w pliku **main.py:**

- **Test1():** Jak algorytmy sobie poradzą z 25, 75, 125 procesami o czasach nadejścia 0 i czasie wykonania od 1 do 10
- **Test2():** Jak algorytmy sobie poradzą z 25, 75, 125 procesów o czasie wykonania z rozkładu dwumianowego o parametrach p = 0.5, n = 2 * średnia. średnia = 10. I czasie nadejścia z przedziału od 0 do 10

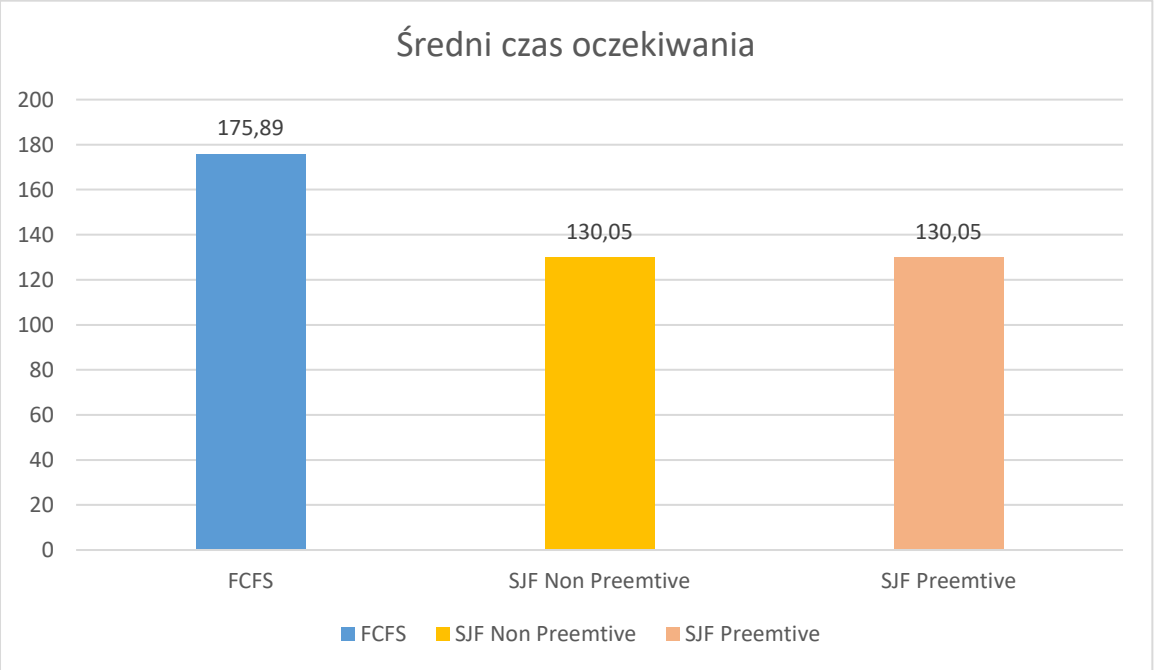
Wyniki eksperymentów

Pliki test1.txt i test2.txt zawierają dane wygenerowanych procesów wraz z wynikami (średnie czasy oczekiwania).

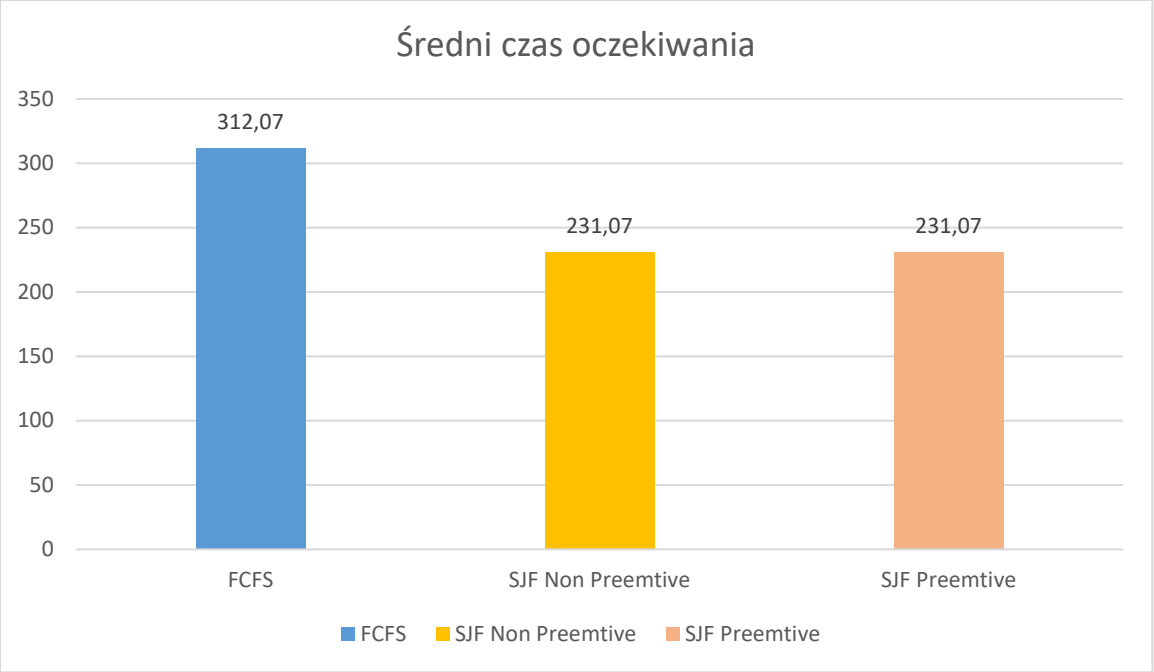
TEST1():



Rysunek 1. Wykres zawiera średnie czasy oczekiwania dla 25 procesów

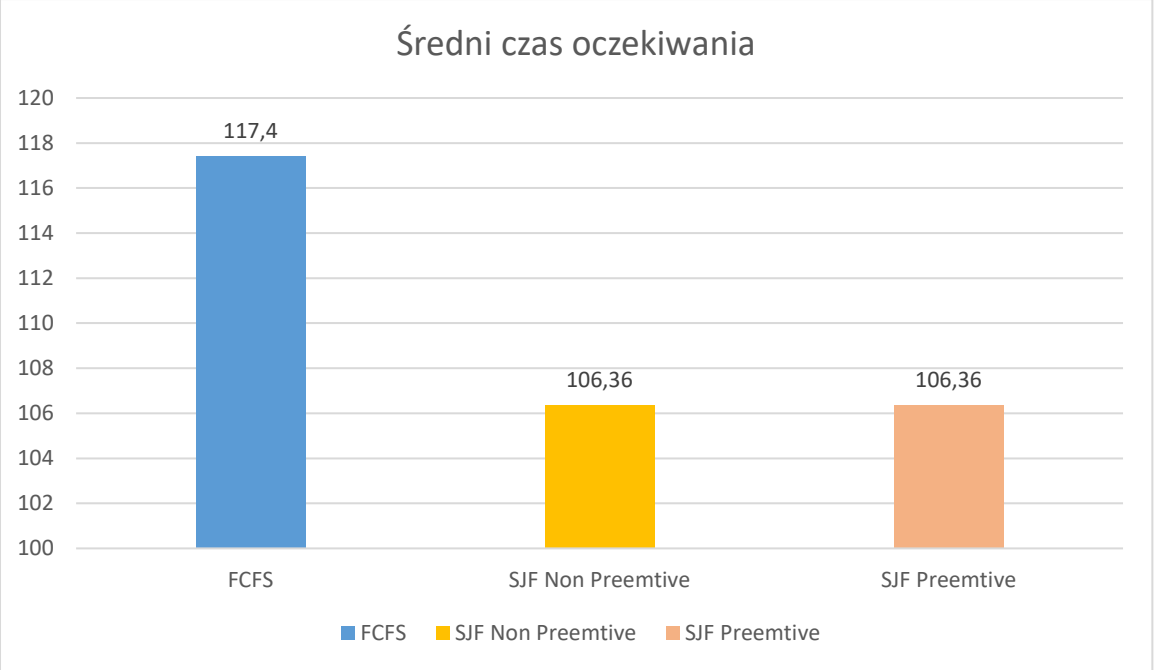


Rysunek 2. Wykres zawiera średnie czasy oczekiwania dla 75 procesów

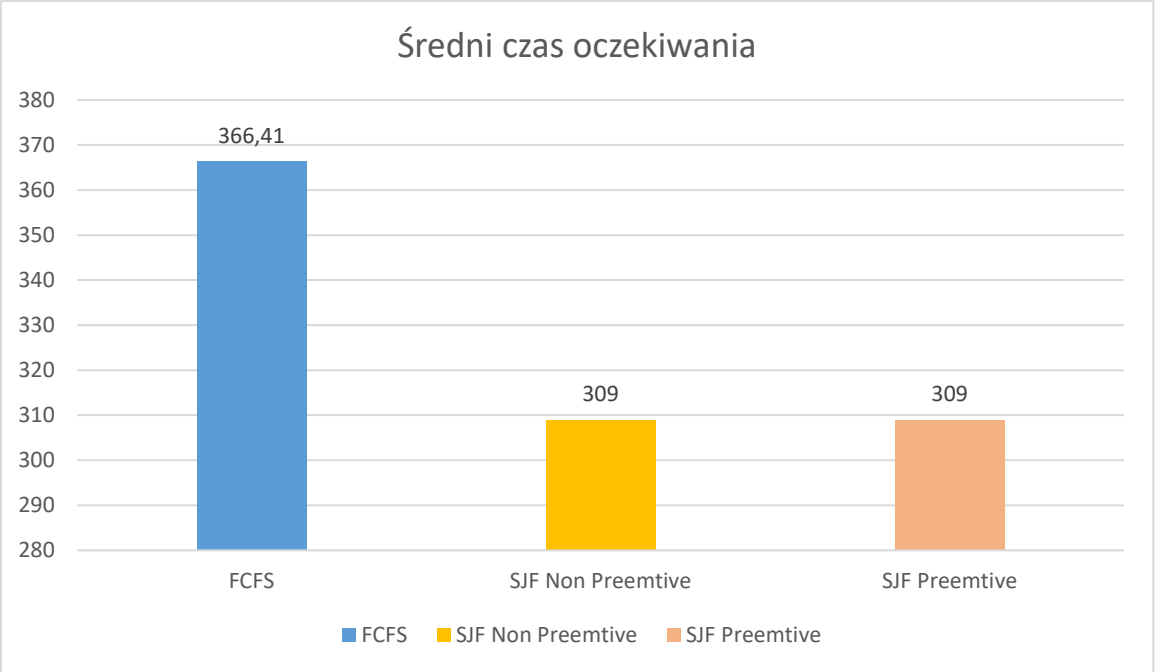


Rysunek 3. Wykres zawiera średnie czasy oczekiwania dla 125 procesów

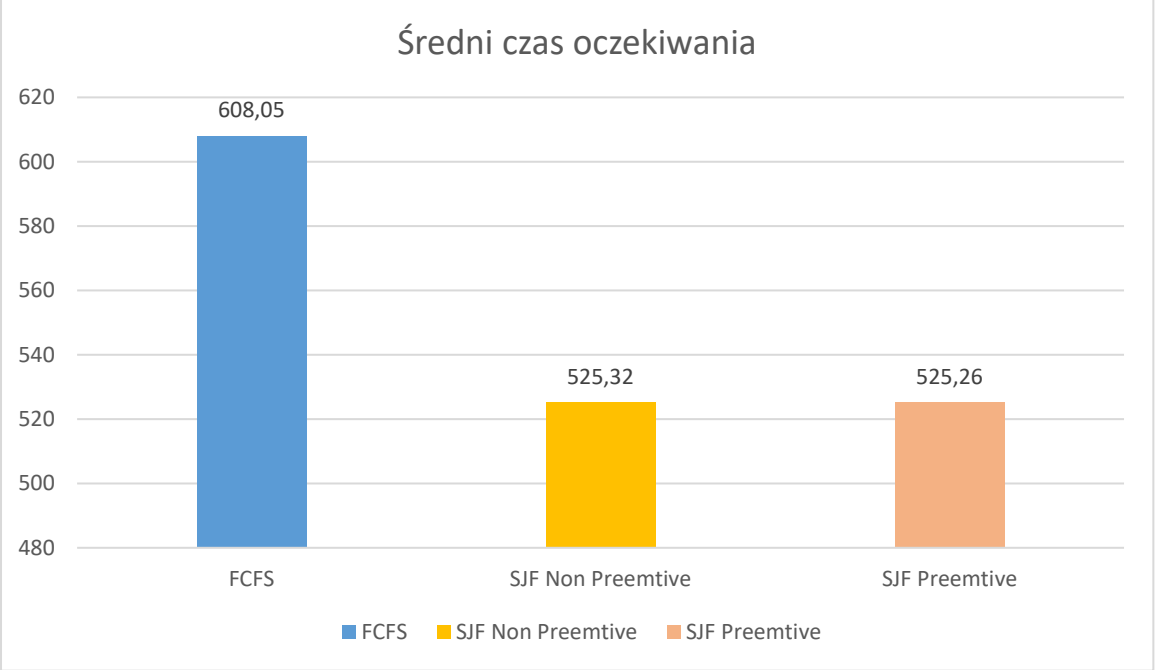
TEST2():



Rysunek 4. Wykres zawiera średnie czasy oczekiwania dla 25 procesów



Rysunek 5. Wykres zawiera średnie czasy oczekiwania dla 75 procesów



Rysunek 6. Wykres zawiera średnie czasy oczekiwania dla 125 procesów

Wnioski

Na podstawie przedstawionych danych dotyczących średnich czasów oczekiwania dla algorytmów FCFS i SJF możemy wyciągnąć kilka wniosków:

FCFS ma wyższy średni czas oczekiwania niż SJF w obu wersjach. Algorytm ten, choć prosty w implementacji, cechuje się jednak wysokim czasem oczekiwania, szczególnie gdy liczba procesów rośnie.

W porównaniu z FCFS, **SJF** radzi sobie lepiej, skracając czas oczekiwania poprzez wykonywanie najkrótszych procesów najpierw. Jest efektywny zarówno dla procesów o dużych, jak i małych czasach trwania.

Warto zauważyć, że wywłaszczająca i niewywłaszczająca wersji SJF mają takie same czasy oczekiwania. Różnica polega tylko na tym, że procesy o bardzo małych czasach wykonania będą czekały mniej, ze „szkodą” procesom o dużych czasach wykonania. Więc średnia oczekiwania pozostaje taka sama.

Algorytmy zastępowania stron

Opis algorytmów

FIFO(First In First Out). Algorytm polega na zastępowaniu ramki pamięci, która była najdawniej załadowana.

LRU(Least Recently Used). Algorytm polega na zastępowaniu ramki pamięci, która była najdawniej używana.

Opis procedury testowania

Implementację poniższych testów zawierają funkcje **test3()** i **test4()** w pliku **main.py:**

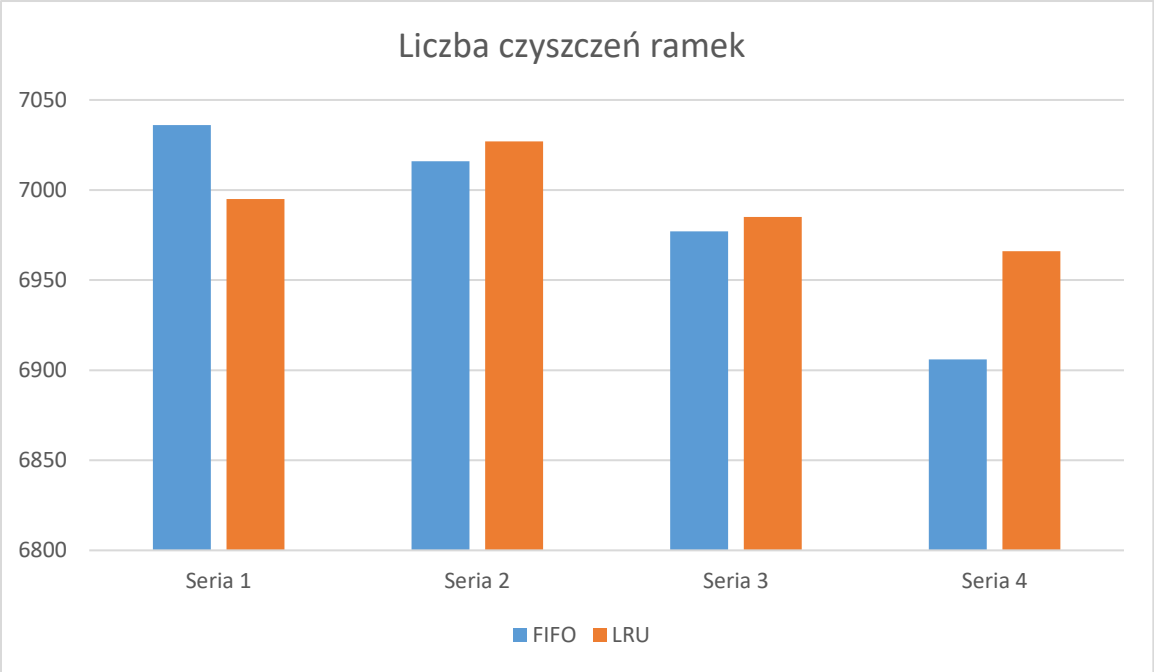
- **Test3():** Jak algorytmy sobie poradzą z 10000-elementową sekwencją odwoływania się do stron z numerami z rozkładu jednostajnego z przedziału (0,10) – to znaczy 10 stron i szerokości ramki pamięci 3.

- **Test4():** Jak algorytmy sobie poradzą z 10000-elementową sekwencją odwoływania się do stron z numerami z rozkładu dwumianowego z przedziału (0,10) – to znaczy 10 stron i szerokości ramki pamięci 7.

  Funkcje zwracają ilość zamian ramek pamięci co służy przedmiotem dla kolejnej analizy. Te liczby zapisane w plikach test3.txt i test4.txt.

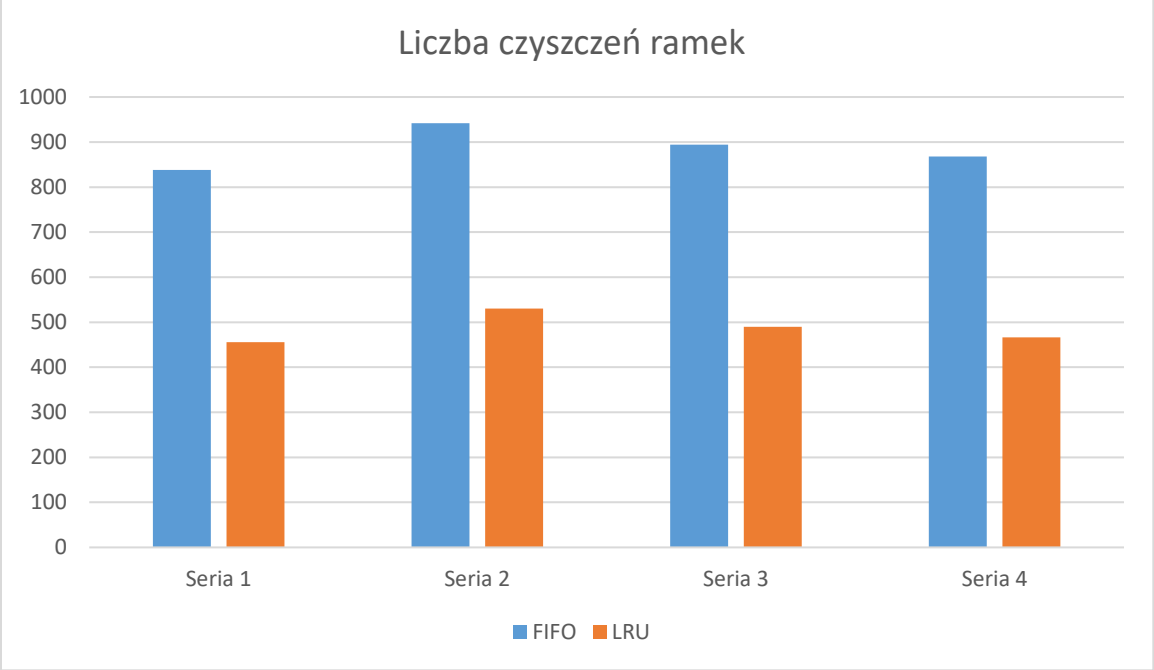
Wyniki eksperymentów

TEST3():



Rysunek 2.1 *10000 odwoływań, sekwencja z rozkładu jednostajnego, szerokość ramki 3*

TEST4():



Rysunek 3.2 *10000 odwoływań, sekwencja z rozkładu dwumianowego, szerokość ramki 7*

Wnioski

Na podstawie przedstawionych danych dotyczących można wyciągnąć kilka wniosków:

Przy generowaniu sekwencji odwoływań za pomocą rozkładu jednostajnego, co znaczy, że strona pojawi się tyle samo raz jak inne, nie można zauważyć różnicy pomiędzy algorytmami FIFO i LRU. Oba algorytmy powodują taką samą liczbę zamian stron.

Sprawa jest inna, gdy sekwencja odwoływań wygenerowana za pomocą rozkładu Bernoulliego, co znaczy, że istnieją strony, do których odwoływań będzie więcej niż do innych. W tym przypadku algorytm LRU wykazuje większą wydajność, ponieważ, jeśli do strony było już odwołanie, ten algorytm oczekuje, że do tej strony będą jeszcze odwołania, i nie kasuje tej strony z ramki pamięci.