Systemy Operacyjne

Programy symulacyjne

Tomasz Świątek - 273142

styczeń 2024

Spis treści

1 Wstęp

Celem projektu jest implementacja czterech wybranych algorytmów i programów symulacyjnych, przeprowadzenie prostych eksperymentów porównujących ich działanie oraz prezentacja uzyskanych rezultatów. Do wykonania była symulacja działania dwóch algorytmów planowania czasu procesora (*Process scheduling*) dla zamkniętej puli zadań. Wybrano:

- FCFS - First Come First Serve
- SJF - Shortest Job First

Konieczna była do przeprowadzenia także symulacja działania algorytmów zastępowania stron (*Page replacement*), a wybrano:

- FIFO - First In First Out
- LRU - Least Recently Used

2 Opis algorytmów

2.1 Algorytmy planowania czasu procesora (Process scheduling)

2.1.1 FCFS

FCFS jest uważany za najprostszy ze wszystkich algorytmów planowania czasu procesora. Polega on na tym, że proces, który jako pierwszy zażąda procesora, otrzymuje go w pierwszej kolejności. Zadania są zawsze wykonywane na zasadzie "kto pierwszy, ten lepszy". FCFS jest łatwy w implementacji i użyciu. Algorytm ten nie jest zbyt wydajny pod względem wydajności, a czas oczekiwania jest dość wysoki. FCFS cierpi z powodu efektu konwoju.

2.1.2 SJF

Shortest job first (SJF) to algorytm planowania czasu procesora, który wybiera oczekujący proces z najmniejszym czasem wykonania do wykonania w następnej kolejności. Ta metoda szeregowania może, ale nie musi, być oparta na wywłaszczaniu. Znacznie skraca średni czas oczekiwania innych procesów oczekujących na wykonanie. Algorytm ten ma tę zaletę, że ma minimalny średni czas oczekiwania spośród wszystkich algorytmów szeregowania procesów systemu operacyjnego. SJF jest zwykle używany do planowania długoterminowego. Jedną z wad SJF jest głód. Wielokrotnie skomplikowane staje się przewidzenie długości nadchodzącego żądania procesora.

2.2 Algorytmy zastępowania stron (Page replacement)

2.3 FIFO

Jest to najprostszy algorytm zastępowania stron. W tym algorytmie system operacyjny śledzi wszystkie strony w pamięci w kolejce, a najstarsza strona znajduje się na początku kolejki. Gdy strona wymaga wymiany, strona z przodu kolejki jest wybierana do usunięcia. FIFO generuje dużą liczbę błędów stron, a dodatkowo obciążony jest *anomalią Belady'ego*, tzn. wraz ze wzrostem liczby ramek może wzrastać liczba błędów stron.
Złożoność czasowa: *O(n)*, gdzie n to liczba stron.
Złożoność przestrzenna: *O(capacity)*, gdzie capacity to liczba ramek

2.4 LRU

W tym algorytmie wymieniana zostaje ta strona, która najdawniej była używana. Używając języka potocznego, można napisać, że algorytm LRU „patrzy w przeszłość". Algorytm LRU jest najpopularniejszym algorytmem stosowanym do wymiany stron.
Złożoność czasowa : *O(n)*, ponieważ wykonuje stałą ilość pracy dla każdego żądania strony.
Złożoność przestrzenna: *O(n+cap)*, gdzie n to rozmiar tablicy wejściowej stron, a cap to liczba ramek.

3 Dane do analizy algorytmów

3.1 Process scheduling

Do porównania wydajności algorytmów planowania czasu procesora wykorzystane zostają następujące dane zebrane dla algorytmów FCFS i SJF:

- Średni czas oczekiwania procesu - *Average waiting time*
- Średni czas realizacji procesu - *Average turnaround time*

3.2 Page replacement

Do porównania wydajności algorytmów zastępowania stron wykorzystane zostają następujące dane zebrane dla algorytmów FIFO i LRU:

- Ilośc wykonanych podmian strony - *Page faults*

4 Struktura programu

- `main.py`: Główny skrypt uruchamiający symulację.
- `data_handler.py`: Moduł zawierający obsługę danych (odczyt z wejściowego json i zapis do wyjściowego json).
- `process_scheduling.py`: Moduł zawierający implementacje algorytmów planowania czasu procesora (FCFS, SJF).
- `page_replacement.py`: Moduł zawierający implementacje algorytmów wymiany stron (FIFO, LRU).
- `gen/process_data_generator.py`: Moduł zawierający generator danych dla procesów.
- `gen/page_data_generator.py`: Moduł zawierający generator danych dla stron.
- `plots/process_results_plotter.py`: Moduł zawierający generator wykresów dla wyników algorytmów planowania czasu procesora.
- `plots/page_results_plotter.py`: Moduł zawierający generator wykresów dla wyników algorytmów zamiany stron.
- `log.py`: Moduł zawierający rejestrator zbierający dane końcowe.
- `page.py`: Klasa strony.
- `process.py`: Klasa procesu.
- `ticker.py`: Klasa wskaźnika czasu.
- `input/`: Katalog zawierający wejściowe pliki json z wygenerowanymi danymi dla procesów i stron.
- `output/`: Katalog zawierający wyniki algorytmów planowania czas procesora i zastępowania stron (surowe dane w json i wykresy).

5 Testowanie algorytmów

W programie do symulacji możliwe jest **generowanie danych** - procesów (dla algorytmów planowania czasu procesora) oraz stron i rozmiarów ramek (dla algorytmów zastępowania stron). Następnie możliwe jest **przeprowadzenie symulacji** (*process scheduling/page replacement*) na wygenerowanych losowych danych, które zapisywane są w formacie *json*. Po wykonaniu algorytmów, **rezultaty są eksportowane** do pliku także w formacie *json*, a następnie wykonane są wykresy dla danych końcowych. Wykresy są zapisywane jako obraz w formacie *png*.

6 Uzyskane rezultaty

6.1 Process scheduling

Przeprowadzono 2 eksperymenty dla wygenerowanych losowo:

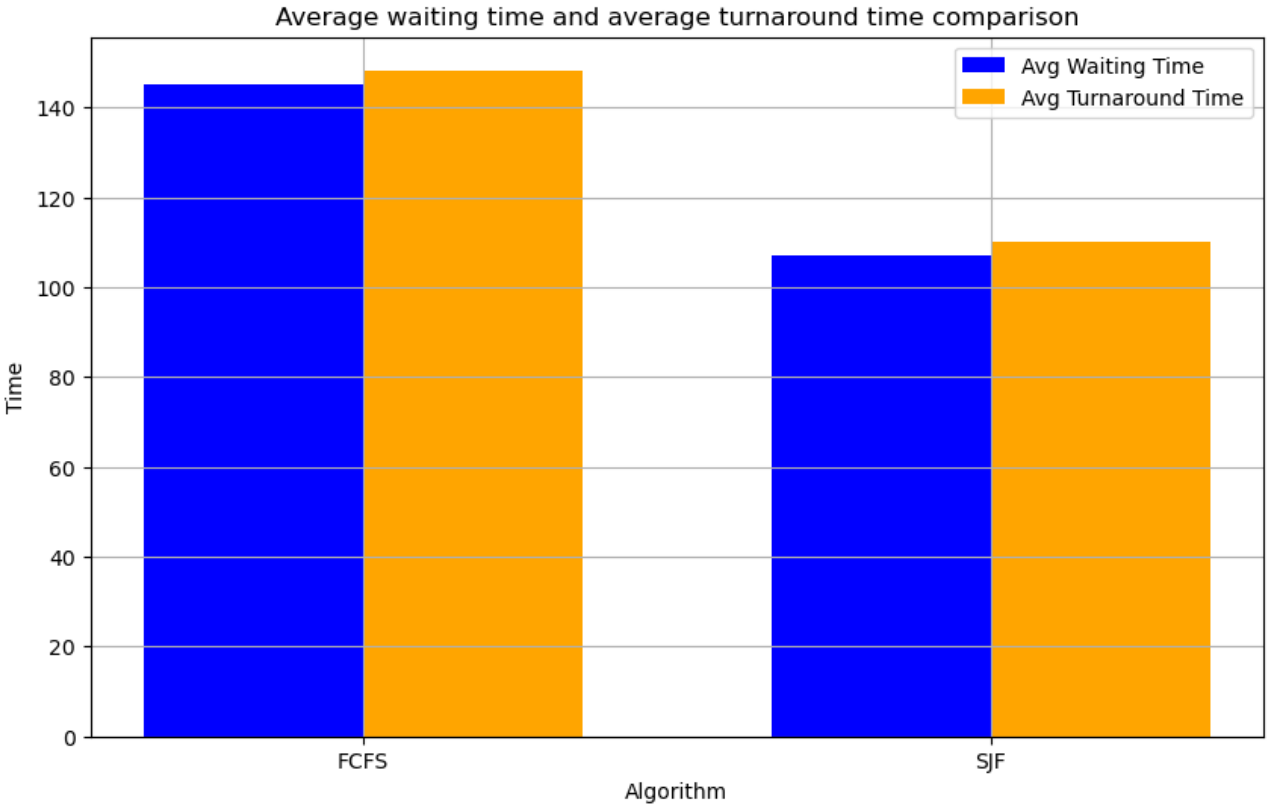
1. 100 procesów o max czasie nadejścia 100 i max czasie wykonania 5
2. 50 procesów o max czasie nadejścia 50 i max czasie wykonania 10

6.1.1 1. eksperyment process scheduling

100 procesów o max czasie nadejścia 100 i max czasie wykonania 5

Tabela 1: Wyniki 1. eksperymentu planowania czasu procesora

Algorytm	Średni czas oczekiwania	Średni czas realizacji
FCFS	145.26	148.26
SJF	107.08	110.08



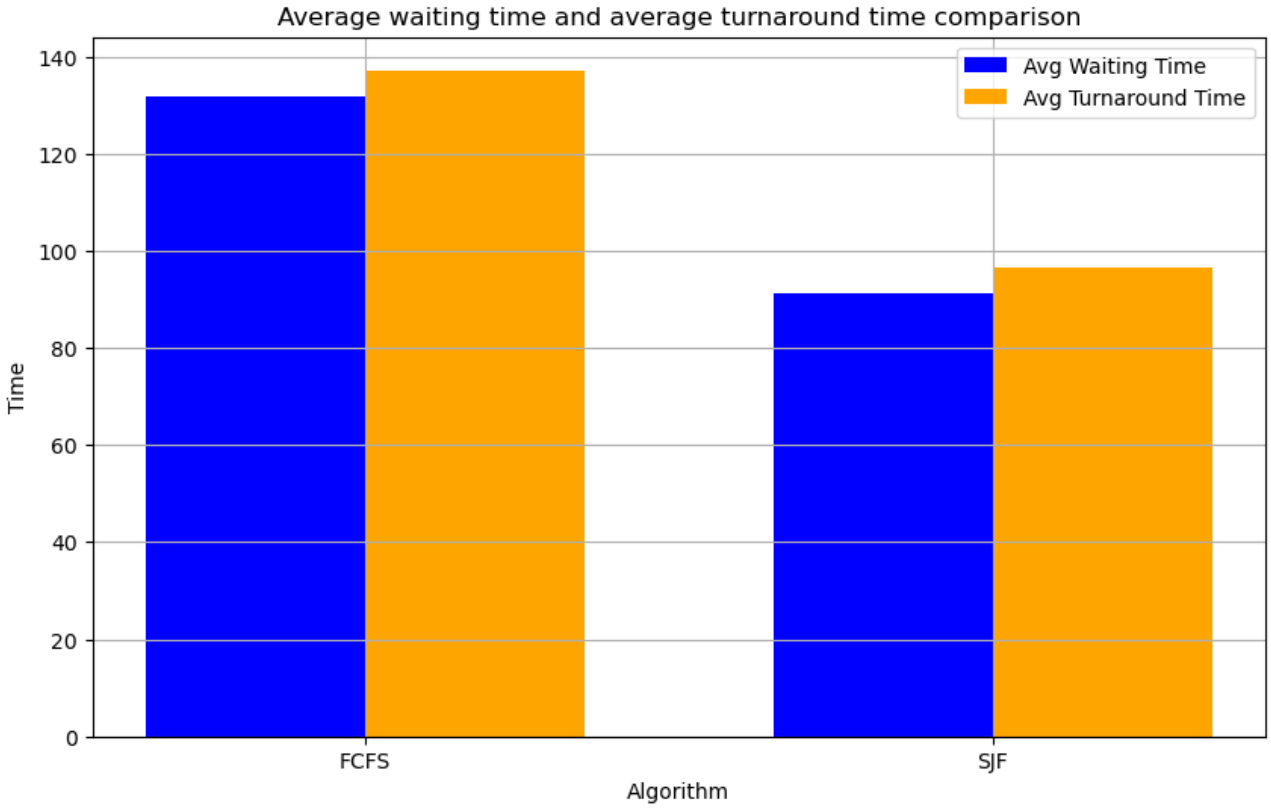
Wykres 1: Średni czas oczekiwania i średni czas realizacji procesów dla 1. eksperymentu

6.1.2 2. eksperyment process scheduling

50 procesów o max czasie nadejścia 50 i max czasie wykonania 10

Tabela 2: Wyniki 2. eksperymentu planowania czasu procesora

Algorytm	Średni czas oczekiwania	Średni czas realizacji
FCFS	131.84	137.14
SJF	91.12	96.42



Wykres 2: Średni czas oczekiwania i średni czas realizacji procesów dla 2. eksperymentu

6.1.3 Analiza

SJF cechuje sie mniejszym średnim czasem oczekiwania procesów oraz mniejszym średnim czasem realizacji procesów dla przeprowadzonych eksperymentów.

6.2 Page replacement

Przeprowadzono 3 eksperymenty dla wygenerowanych losowo:

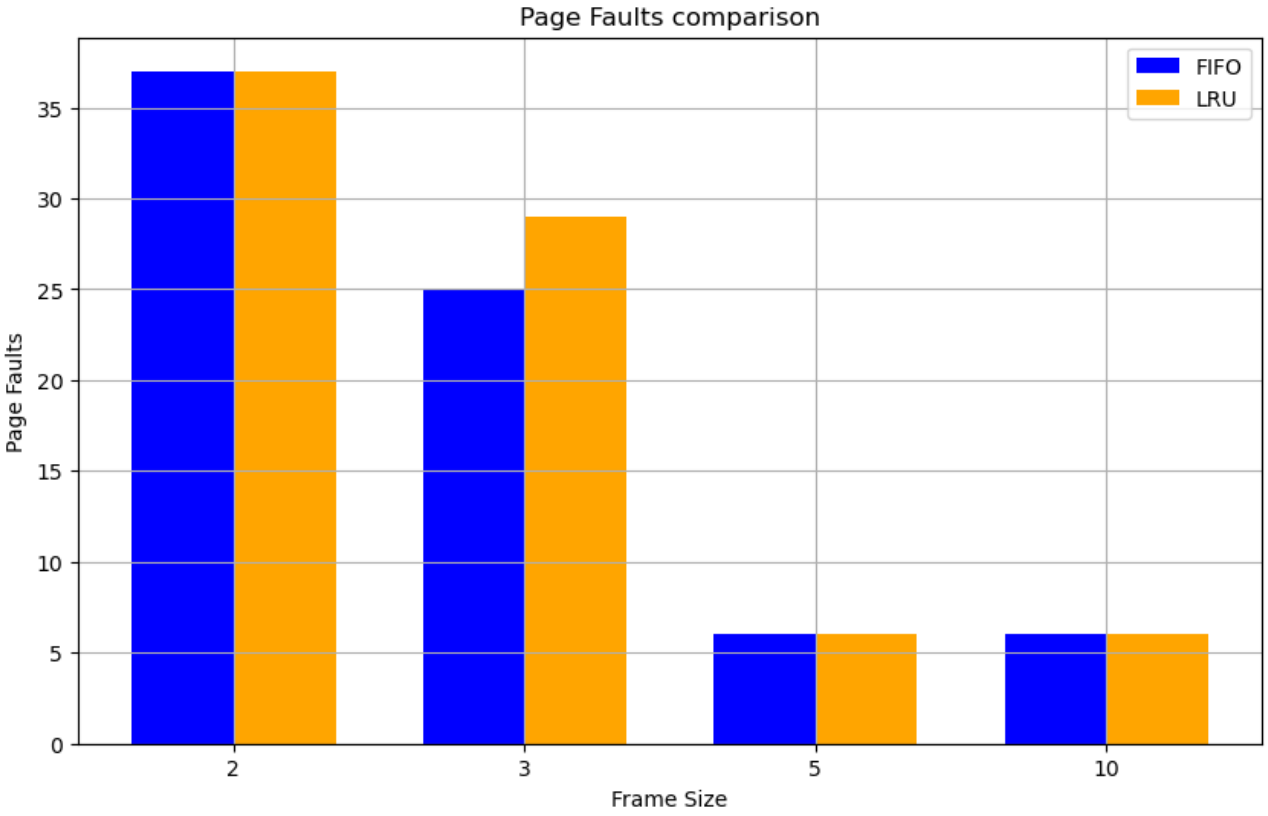
1. 50 stron o zakresie 5 dla 4 ramek o rozmiarach: 2, 3, 5, 10
2. 50 stron o zakresie 20 dla 4 ramek o rozmiarach: 2, 3, 5, 10
3. 100 stron o zakresie 50 dla 5 ramek o rozmiarach: 3, 5, 10, 20, 30

6.2.1 1. eksperyment page replacement

50 stron o zakresie 5 dla 4 ramek o rozmiarach: 2, 3, 5, 10

Tabela 3: Wyniki 1. eksperymentu zastępowania stron

Algorytm	Rozmiar ramki	Liczba wymian
FIFO	2	37
	3	25
	5	6
	10	6
LRU	2	37
	3	29
	5	6
	10	6



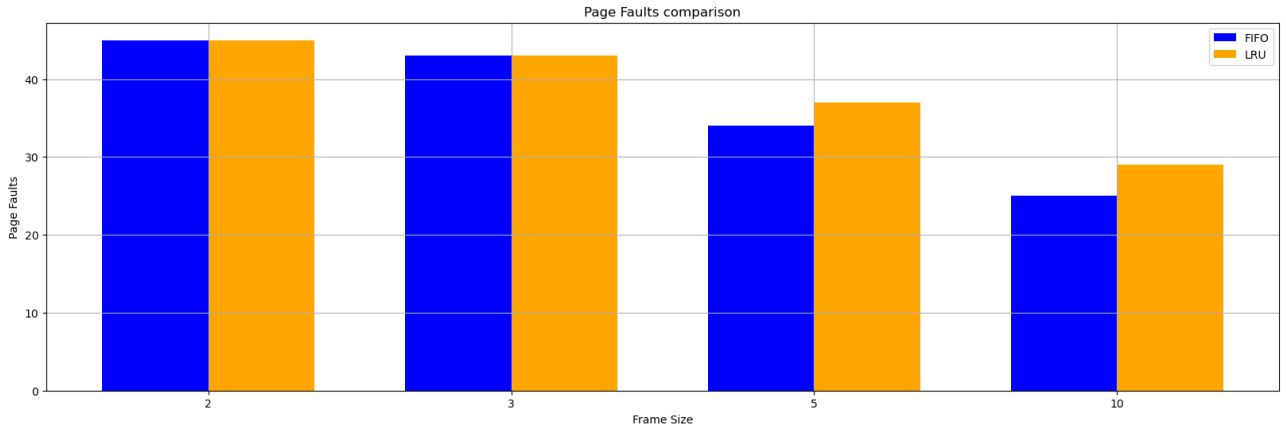
Wykres 3: Ilość zastąpień stron *page faults* dla różnych wielkości ramki dla 1. eksperymentu

6.2.2 2. eksperyment page replacement

50 stron o zakresie 20 dla 4 ramek o rozmiarach: 2, 3, 5, 10

Tabela 4: Wyniki 2. eksperymentu planowania czasu procesora

Algorytm	Rozmiar ramki	Liczba wymian
FIFO	2	45
	3	43
	5	34
	10	25
LRU	2	45
	3	43
	5	37
	10	29



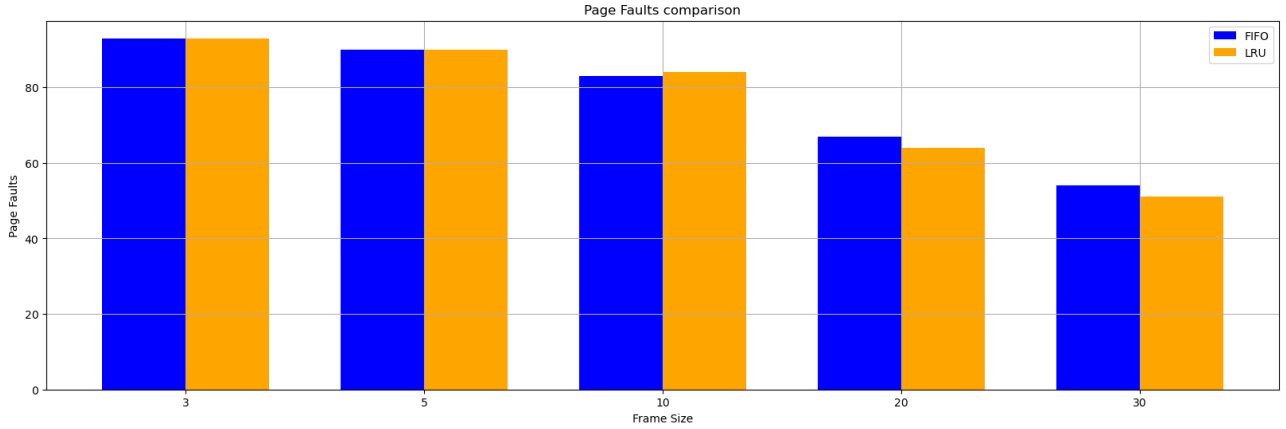
Wykres 4: Ilość zastąpień stron *page faults* dla różnych wielkości ramki dla 2. eksperymentu

6.2.3 3. eksperyment page replacement

100 stron o zakresie 50 dla 5 ramek o rozmiarach: 3, 5, 10, 20, 30

Tabela 5: Wyniki 3. eksperymentu planowania czasu procesora

Algorytm	Rozmiar ramki	Liczba wymian
FIFO	3	93
	5	90
	10	83
	20	67
	30	54
LRU	3	93
	5	90
	10	84
	20	64
	30	51



Wykres 5: Ilość zastąpień stron *page faults* dla różnych wielkości ramki dla 3. eksperymentu

6.2.4 Analiza

FIFO - Dla dużej ilości stron i małego rozmiaru ramki, algorytm ten może prowadzić do większej liczby zamian (*page faults*), co wpływa na wydajność systemu. Im większy rozmiar ramki tym algorytm LRU radzi sobie lepiej. LRU zazwyczaj skutkuje mniejszą ilością zamian stron niż FIFO, szczególnie dla dużego zakresu numerów stron i większych rozmiarów ramek.

7 Wnioski

Wybór algorytmów planowania czasu procesora i zamiany stron zależy od konkretnych warunków systemowych, takich jak liczba procesów, zakres czasu wykonania, ilość dostępnej pamięci RAM, czy też charakterystyka dostępnych stron i rozmiar ramki. Optymalny wybór algorytmu zależy także od specyfiki aplikacji i jej wymagań. Algorytmy, które sprawdzają się dobrze w jednym środowisku, mogą nie być równie efektywne w innym. Podsumowując, przeprowadzone symulacje pozwalają zobaczyć, jak różne czynniki wpływają na działanie algorytmów, a ich analiza umożliwia wybór optymalnych strategii planowania czasu procesora oraz zarządzania pamięcią w konkretnych scenariuszach.