Autor: Tomasz Trefler

**Spis treści**

[Opis algorytmów 2](#_Toc168589365)

[GENERATOR\_PROCESÓW 2](#_Toc168589366)

[GENERATOR\_SEKWENCJI 3](#_Toc168589367)

[FCFS 4](#_Toc168589368)

[ROUNDROBIN 6](#_Toc168589369)

[FIFO 8](#_Toc168589370)

[LRU 9](#_Toc168589371)

[Opis procedury testowania algorytmów 10](#_Toc168589372)

[Testowanie algorytmów zarządzania procesami 10](#_Toc168589373)

[Testowanie algorytmów zastępowania stron 12](#_Toc168589374)

[Opracowane wyniki eksperymentów 14](#_Toc168589375)

[Wyniki testowania algorytmów zarządzania procesami 14](#_Toc168589376)

[Wyniki testowania algorytmów zastępowania stron 16](#_Toc168589377)

[Wnioski 19](#_Toc168589378)

[Alogrytmy zarządzania procesami . 19](#_Toc168589379)

[Alogrytmy zastępowania stron 19](#_Toc168589380)

[Końcowe wnioski 20](#_Toc168589381)

# Opis algorytmów

## GENERATOR\_PROCESÓW

Celem kodu jest wygenerowanie danych dotyczących procesów i zapisanie ich do pliku CSV o nazwie PROCESY.csv. Każdy proces jest reprezentowany przez identyfikator (PID), czas przybycia (Arrival) oraz czas wykonania (Execution). Kod umożliwia użytkownikowi określenie liczby procesów, które mają zostać wygenerowane.

#### Opis Kodu

##### Funkcja main(n)

Funkcja main(n) jest główną funkcją programu odpowiedzialną za generowanie i zapis danych procesów do pliku CSV. Przyjmuje ona jeden argument n, który określa liczbę procesów do wygenerowania.

* **Blok try**:
  + Otwarcie pliku PROCESY.csv do zapisu. Jeśli plik nie istnieje, zostanie utworzony nowy.
  + Zapis nagłówków kolumn do pliku (PID, Arrival, Execution).
* **Pętla for**:
  + Iteracja przez zakres od 1 do n (włącznie) w celu wygenerowania danych dla każdego procesu.
  + Generowanie czasu przybycia arrival\_time
  + Generowanie czasu wykonania execution\_time
  + Zapisanie danych procesu do pliku.
* **Blok except**:
  + Obsługa wyjątków na wypadek niepowodzenia w zapisie do pliku, wyświetlenie komunikatu o błędzie.
* **Komunikat o powodzeniu**:
  + Po zakończeniu zapisu wyświetlenie komunikatu informującego o pomyślnym zapisaniu danych do pliku.

##### Blok if \_\_name\_\_ == "\_\_main\_\_":

* Pobiera liczbę procesów od użytkownika za pomocą funkcji input().
* Wywołuje funkcję main(n) z podaną liczbą procesów.

## GENERATOR\_SEKWENCJI

Celem kodu jest wygenerowanie sekwencji stron i zapisanie ich do pliku CSV o nazwie SEKWENCJE.csv. Każda sekwencja składa się z losowej liczby, reprezentującej stronę. Użytkownik określa liczbę stron, które mają zostać wygenerowane.

#### Opis Kodu

#### Funkcja main(n)

Funkcja main(n) jest główną funkcją programu odpowiedzialną za generowanie i zapis sekwencji stron do pliku CSV. Przyjmuje ona jeden argument n, który określa liczbę stron do wygenerowania.

* **Blok try:**
  + Otwarcie pliku SEKWENCJE.csv do zapisu. Jeśli plik nie istnieje, zostanie utworzony nowy.
  + Utworzenie obiektu csv.writer, który będzie używany do zapisywania danych do pliku.
* **Pętla for:**
  + Iteracja przez zakres od 0 do n-1 w celu wygenerowania danych dla każdej strony.
  + Generowanie losowej liczby, reprezentującej stronę.
  + Zapisanie danych strony do pliku.
* **Blok except:**
  + Obsługa wyjątków na wypadek niepowodzenia w zapisie do pliku, wyświetlenie komunikatu o błędzie.
* **Komunikat o powodzeniu:**
  + Po zakończeniu zapisu wyświetlenie komunikatu informującego o pomyślnym zapisaniu danych do pliku.

#### Blok if \_\_name\_\_ == "\_\_main\_\_":

* Pobiera liczbę stron od użytkownika za pomocą funkcji input().
* Wywołuje funkcję main(n) z podaną liczbą stron.

## FCFS

Celem kodu jest symulacja działania algorytmu szeregowania procesów "First-Come, First-Served" (FCFS). Algorytm ten jest jednym z najprostszych metod zarządzania procesami w systemach operacyjnych, przydzielając czas procesora procesom w kolejności ich przybycia. Kod realizuje: odczyt danych procesów z pliku CSV, obliczenia istotnych czasów dla każdego procesu, a następnie zapis wyników do pliku CSV.

#### Opis Kodu

##### Klasa Process

Klasa Process reprezentuje pojedynczy proces z następującymi atrybutami:

* pid: identyfikator procesu
* arrival\_time: czas przybycia procesu
* execution\_time: czas wykonywania się procesu
* start\_time: czas rozpoczęcia wykonywania się procesu
* completion\_time: czas zakończenia wykonywania się procesu
* turnaround\_time: czas od przybycia do zakończenia procesu
* waiting\_time: czas oczekiwania na rozpoczęcie wykonywania się procesu

##### Funkcja calculate\_times

Funkcja calculate\_times oblicza kluczowe czasy dla listy procesów:

* current\_time: bieżący czas w symulacji
* completed\_processes: lista zakończonych procesów

Funkcja działa w pętli do momentu przetworzenia wszystkich procesów:

* Wybiera procesy dostępne do wykonania na podstawie bieżącego czasu.
* Spośród dostępnych procesów wybiera ten, który przybył najwcześniej.
* Ustawia czasy rozpoczęcia, zakończenia, oczekiwania oraz obrotu dla wybranego procesu.
* Aktualizuje bieżący czas do czasu zakończenia wybranego procesu.
* Dodaje zakończony proces do listy zakończonych procesów i usuwa go z listy procesów do przetworzenia.
* Jeśli nie ma dostępnych procesów, przesuwa bieżący czas do przodu.

#### Funkcja save\_to\_csv

Funkcja save\_to\_csv zapisuje listę procesów do pliku CSV:

* Tworzy plik FCFS.csv i zapisuje nagłówki kolumn.
* Iteruje przez listę procesów, zapisując ich atrybuty do pliku.
* Obsługuje wyjątki i informuje użytkownika o ewentualnych problemach z zapisem.

#### Funkcja calculate\_average\_times

Funkcja calculate\_average\_times oblicza średnie czasy obrotu i oczekiwania dla podanej listy procesów:

* Sumuje czasy obrotu i oczekiwania dla wszystkich procesów.
* Oblicza średnie czasy na podstawie liczby procesów.

#### Funkcja main

Funkcja main jest punktem wejścia do programu:

* Odczytuje plik PROCESY.csv z danymi procesów.
* Tworzy listę obiektów Process na podstawie danych z pliku.
* Oblicza czasy dla procesów przy użyciu funkcji calculate\_times.
* Oblicza średnie czasy obrotu i oczekiwania za pomocą funkcji calculate\_average\_times.
* Zapisuje wyniki do pliku FCFS.csv za pomocą funkcji save\_to\_csv.

## ROUNDROBIN

Celem kodu jest implementacja i symulacja algorytmu szeregowania procesów Round Robin (RR), który jest powszechnie stosowany w systemach operacyjnych. Algorytm ten przydziela procesom czas CPU w równych, stałych jednostkach zwanych kwantami czasu. Kod zawiera pełny cykl odczytu danych procesów z pliku CSV, obliczenia istotnych czasów dla każdego procesu, aż po zapis wyników do pliku CSV.

#### Opis Kodu

##### Klasa Process

Klasa Process reprezentuje pojedynczy proces z następującymi atrybutami:

* pid: identyfikator procesu
* arrival\_time: czas przybycia procesu
* execution\_time: czas wykonywania się procesu
* remaining\_time: pozostały czas wykonywania się procesu
* completion\_time: czas zakończenia wykonywania się procesu
* waiting\_time: łaczny czas jaki proces oczekiwał gdy nie wykonywał się
* start\_time: czas rozpoczęcia wykonywania się procesu
* turnaround\_time: czas od przybycia do zakończenia procesu

#### Funkcja round\_robin

Funkcja round\_robin implementuje algorytm Round Robin dla listy procesów z podanym kwantem czasu:

* current\_time: bieżący czas w symulacji
* completed\_processes: lista zakończonych procesów

Funkcja działa w pętli, dopóki nie zostaną przetworzone wszystkie procesy:

* Wybiera procesy dostępne do wykonania na podstawie bieżącego czasu.
* Dodaje procesy do kolejki na podstawie ich czasu przybycia.
* Wybiera i przetwarza procesy zgodnie z kwantem czasu.
* Aktualizuje czasy dla wybranego procesu.
* Jeśli proces może zostać zakończony w bieżącym kwancie czasu, jego czas wykonania jest zerowany, a czasy zakończenia, oczekiwania i obrotu są obliczane.
* Jeśli proces nie może zostać zakończony w bieżącym kwancie czasu, jego pozostały czas jest zmniejszany o kwant czasu, a proces jest dodawany z powrotem do kolejki.
* Jeśli nie ma dostępnych procesów, bieżący czas jest inkrementowany.

#### Funkcja save\_to\_csv

Funkcja save\_to\_csv zapisuje listę procesów do pliku CSV:

* Tworzy plik ROUNDROBIN.csv i zapisuje nagłówki kolumn.
* Iteruje przez listę procesów, zapisując ich atrybuty do pliku.
* Obsługuje wyjątki i informuje użytkownika o ewentualnych problemach z zapisem.

#### Funkcja calculate\_average\_times

Funkcja calculate\_average\_times oblicza średnie czasy obrotu i oczekiwania dla podanej listy procesów:

* Sumuje czasy obrotu i oczekiwania dla wszystkich procesów.
* Oblicza średnie czasy na podstawie liczby procesów.

#### Funkcja main

Funkcja main jest punktem wejścia do programu:

* Odczytuje plik PROCESY.csv z danymi procesów.
* Tworzy listę obiektów Process na podstawie danych z pliku.
* Oblicza czasy dla procesów przy użyciu funkcji round\_robin.
* Oblicza średnie czasy obrotu i oczekiwania za pomocą funkcji calculate\_average\_times.
* Zapisuje wyniki do pliku ROUNDROBIN.csv za pomocą funkcji save\_to\_csv.

## FIFO

Celem kodu jest implementacja i symulacja algorytmu zamiany stron FIFO (First-In-First-Out). Algorytm ten jest prostą metodą zarządzania stronami w pamięci, gdzie strony są usuwane w kolejności, w jakiej zostały załadowane. Algorytm implementuje pełny proces zarządzania stronami, odczytując dostępne strony, kontrolując kolejkę stron w pamięci i pokazując aktualny stan pamięci po każdym dostępie.

#### Opis Kodu

##### Klasa FIFO

Klasa FIFO reprezentuje strukturę zarządzającą stronami w pamięci przy użyciu algorytmu FIFO. Posiada następujące atrybuty i metody:

* **Atrybuty**:
  + capacity: maksymalna liczba stron, które mogą być jednocześnie przechowywane w pamięci.
  + queue: lista reprezentująca kolejkę stron w pamięci, w której najstarsze strony są usuwane jako pierwsze.
  + cache: zbiór stron w pamięci, który umożliwia szybkie sprawdzanie, czy strona znajduje się już w pamięci.
* **Metody**:
  + \_\_init\_\_(self, capacity: int): inicjalizuje obiekt FIFO z określoną pojemnością, tworząc pustą kolejkę i zbiór stron w pamięci.
  + access\_page(self, page): zarządza dostępem do strony:
    - Jeśli strona nie znajduje się w pamięci (page fault), usuwa najstarszą stronę z kolejki, jeśli jest pełna, i dodaje nową stronę do kolejki i pamięci.
    - Jeśli strona znajduje się w pamięci (page hit), informuje o dostępie do strony bez konieczności jej dodawania.
  + display(self): wyświetla bieżące strony znajdujące się w pamięci.

##### Funkcja main

Funkcja main jest punktem wejścia do programu i wykonuje następujące czynności:

* Tworzy obiekt FIFO o pojemności zadanej przez użytkownika.
* Odczytuje plik SEKWENCJE.csv z danymi procesów.
* Tworzy listę obiektów pages na podstawie danych z pliku
* Iteruje przez listę stron, symulując dostęp do każdej strony przy użyciu metody access\_page oraz wyświetla aktualny stan pamięci po każdym dostępie za pomocą metody display.

## LRU

Celem kodu jest implementacja i symulacja algorytmu zamiany stron LRU (Least Recently Used). Algorytm ten zarządza stronami w pamięci, usuwając te, które były najdłużej nieużywane, aby zrobić miejsce na nowe strony. Algorytm implementuje pełny proces zarządzania stronami, odczytując dostępne strony, kontrolując kolejkę stron w pamięci i pokazując aktualny stan pamięci po każdym dostępie.

#### Opis Kodu

##### Klasa LRU

Klasa LRU reprezentuje strukturę zarządzającą stronami w pamięci przy użyciu algorytmu LRU. Posiada następujące atrybuty i metody:

* **Atrybuty**:
  + capacity: maksymalna liczba stron, które mogą być jednocześnie przechowywane w pamięci.
  + cache: lista reprezentująca cache stron w pamięci, w której najdłużej nieużywane strony są usuwane jako pierwsze.
* **Metody**:
  + \_\_init\_\_(self, capacity: int): inicjalizuje obiekt LRU z określoną pojemnością, tworząc pustą listę cache.
  + access\_page(self, page): zarządza dostępem do strony:
    - Jeśli strona nie znajduje się w pamięci (page fault), usuwa najstarszą stronę z cache, jeśli jest pełna, i dodaje nową stronę.
    - Jeśli strona znajduje się w pamięci (page hit), przenosi ją na koniec listy, aby oznaczyć ją jako ostatnio używaną.
  + display(self): wyświetla bieżące strony znajdujące się w pamięci.

##### Funkcja main

Funkcja main jest punktem wejścia do programu i wykonuje następujące czynności:

* Tworzy obiekt LRU o pojemności zadanej przez użytkownika.
* Odczytuje plik SEKWENCJE.csv z danymi procesów.
* Tworzy listę obiektów pages na podstawie danych z pliku
* Iteruje przez listę stron, symulując dostęp do każdej strony przy użyciu metody access\_page oraz wyświetla aktualny stan pamięci po każdym dostępie za pomocą metody display.

# Opis procedury testowania algorytmów

## Testowanie algorytmów zarządzania procesami

Testowanie algorytmów FCFS (First-Come, First-Served) i Round Robin przeprowadziłem wykonując dwa eksperymenty z różnymi liczbami procesów oraz różnymi czasami przybycia i wykonania procesów. Wyniki eksperymentów mogą dostarczyć informacji na temat efektywności algorytmów w różnych scenariuszach.

#### Procedura Testowania

##### Eksperyment 1: Procesy o Losowych Czasach Nadejścia i Stałym Czasie Wykonania

1. **Założenia**:
   * Liczba procesów: 25, 75, 125
   * Czas przybycia: losowy, z przedziału od 0 do 100
   * Czas wykonania: stały, równy 5
2. **Kroki**:
   * Dla każdej z trzech liczb procesów (25, 75, 125):
     1. Wygenerować dane procesów zgodnie z założeniami.
     2. Zastosować algorytm FCFS do przetworzenia procesów.
     3. Zastosować algorytm Round Robin do przetworzenia procesów, z ustalonym kwantem czasu (w moim przypadku było to 2).
     4. Zmierzyć wyniki, takie jak czas zakończenia wszystkich procesów, średni czas oczekiwania, średni czas obrotu.
3. **Cele**:
   * Ocenić, jak algorytmy radzą sobie z różnymi liczbami procesów przy losowych czasach przybycia.
   * Porównać średnie czasy oczekiwania i obrotu dla obu algorytmów.

##### Eksperyment 2: Procesy o Stałym Czasie Nadejścia i Losowym Czasie Wykonania

1. **Założenia**:
   * Liczba procesów: 25, 75, 125
   * Czas przybycia: stały, równy 0
   * Czas wykonania: losowy, z przedziału od 1 do 10
2. **Kroki**:
   * Dla każdej z trzech liczb procesów (25, 75, 125):
     1. Wygenerować dane procesów zgodnie z założeniami.
     2. Zastosować algorytm FCFS do przetworzenia procesów.
     3. Zastosować algorytm Round Robin do przetworzenia procesów, z ustalonym kwantem czasu (w moim przypadku było to 2).
     4. Zmierzyć wyniki, takie jak czas zakończenia wszystkich procesów, średni czas oczekiwania, średni czas obrotu obrotu.
3. **Cele**:
   * Ocenić, jak algorytmy radzą sobie z procesami o stałym czasie nadejścia i różnym czasie wykonania.
   * Porównać średnie czasy oczekiwania i obrotu dla obu algorytmów.

#### Przeprowadzenie Eksperymentów

1. **Generowanie Danych Procesów**:
   * Wygenerowanie odpowiednich plików CSV z danymi procesów dla obu eksperymentów.
2. **Uruchomienie Algorytmów**:
   * Uruchomienie algorytmów FCFS i Round Robin z wygenerowanymi danymi procesów.
3. **Zbieranie Wyników**:
   * Zapis wyników w plikach CSV dla późniejszej analizy.
4. **Analiza Wyników**:
   * Obliczenie średnich czasów oczekiwania i obrotu dla obu algorytmów.
   * Porównanie wyników w celu oceny wydajności algorytmów w różnych scenariuszach.

## Testowanie algorytmów zastępowania stron

Testowanie algorytmów FIFO (First-In-First-Out) oraz LRU (Least Recently Used) przeprowadziłem wykonując dwa eksperymenty. Pierwszy z dwoma różnymi sekwencjami stron oraz różną pojemnością pamięci. Drugi z trzema różnej długości sekwencjami stron oraz różną pojemnością pamięci. Wyniki eksperymentów dostarczają informacji na temat efektywności algorytmów w różnych scenariuszach.

#### Procedura Testowania

#### Eksperyment 1: Sekwencje stron o różnych długościach przy różnych pojemnościach pamięci

1. **Założenia:**
   * Sekwencje stron:
     1. [1, 2, 3, 4, 2, 2, 2, 1, 5, 6, 3],
     2. [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2]
   * Pojemności pamięci: 3 strony, 4 strony
2. **Kroki:**
   * Dla każdej sekwencji stron:
     1. Ustawić pojemność pamięci na 3 strony.
     2. Przetestować algorytm FIFO na sekwencji stron.
     3. Przetestować algorytm LRU na sekwencji stron.
     4. Zanotować liczbę błędów stron (page faults) dla obu algorytmów.
     5. Ustawić pojemność pamięci na 4 strony.
     6. Powtórzyć kroki 2-4 dla nowej pojemności pamięci.
3. **Cele:**
   * Ocenić, jak algorytmy FIFO i LRU radzą sobie z różnymi długościami sekwencji stron przy różnych pojemnościach pamięci.
   * Porównać liczbę błędów stron (page faults) dla obu algorytmów.

#### Eksperyment 2: Losowe sekwencje stron przy różnych pojemnościach pamięci

1. **Założenia:**
   * Sekwencje stron: Losowe sekwencje o długości 10, 50, 100 stron (generowane dynamicznie).
   * Pojemności pamięci: 3 strony, 5 stron
2. **Kroki:**
   * Dla każdej długości sekwencji stron:
     1. Wygenerować losową sekwencję stron o podanej długości.
     2. Ustawić pojemność pamięci na 3 strony.
     3. Przetestować algorytm FIFO na sekwencji stron.
     4. Przetestować algorytm LRU na sekwencji stron.
     5. Zanotować liczbę błędów stron (page faults) dla obu algorytmów.
     6. Ustawić pojemność pamięci na 5 stron.
     7. Powtórzyć kroki 3-5 dla nowej pojemności pamięci.
3. **Cele:**
   * Ocenić, jak algorytmy FIFO i LRU radzą sobie z losowymi sekwencjami stron o różnej długości przy różnych pojemnościach pamięci.
   * Porównać liczbę błędów stron (page faults) dla obu algorytmów.

#### Przeprowadzenie Eksperymentów

1. **Generowanie Sekwencji Stron:**
   * Wygenerowanie odpowiednich sekwencji stron zgodnie z założeniami eksperymentów.
2. **Uruchomienie Algorytmów:**
   * Uruchomienie algorytmów FIFO i LRU na wygenerowanych sekwencjach stron.
3. **Zbieranie Wyników:**
   * Zapis wyników, tj. liczby błędów stron (page faults) dla obu algorytmów przy różnych sekwencjach stron i pojemnościach pamięci.
4. **Analiza Wyników:**
   * Porównanie liczby błędów stron (page faults) dla algorytmów FIFO i LRU.
   * Wyciągnięcie wniosków na temat efektywności obu algorytmów w różnych scenariuszach.

# Opracowane wyniki eksperymentów

## Wyniki testowania algorytmów zarządzania procesami

##### Eksperyment 1: Procesy o Losowych Czasach Nadejścia i Stałym Czasie Wykonania

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LICZBA PROCESÓW | ROUND ROBIN | | FCFS | |
| TURNAROUND | WAITING | TURNAROUND | WAITING |
| 25 | 41.84 | 36.84 | 26.4 | 21.4 |
| 75 | 251.693 | 246.693 | 145.346 | 140.346 |
| 125 | 467.272 | 462.272 | 265.864 | 260.864 |

*Tabela 1: Średnie czasy oczekiwania i obrotu*

**Gdzie:**

* TURNAROUND – średni czas od przybycia do zakończenia danej liczby procesów
* WAITING – średni czas oczekiwania danej liczby procesów gdy się nie wykonywały

**Obraz zawierający tekst, linia, zrzut ekranu, diagram

Opis wygenerowany automatycznie**

*Wykres 1: Porównanie średnich czasów oczekiwania i obrotu*

##### Eksperyment 2: Procesy o Stałym Czasie Nadejścia i Losowym Czasie Wykonania

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LICZBA PROCESÓW | ROUND ROBIN | | FCFS | |
| TURNAROUND | WAITING | TURNAROUND | WAITING |
| 25 | 74.0 | 69.24 | 66.24 | 61.48 |
| 75 | 269.48 | 264.0 | 208.72 | 203.24 |
| 125 | 373.832 | 368.848 | 334.128 | 329.144 |

*Tabela 2: Średnie czasy oczekiwania i obrotu*

**Gdzie:**

* TURNAROUND – średni czas od przybycia do zakończenia danej liczby procesów
* WAITING – średni czas oczekiwania danej liczby procesów gdy się nie wykonywały

*Obraz zawierający tekst, linia, diagram, zrzut ekranu

Opis wygenerowany automatycznie*

*Wykres 2: Porównanie średnich czasów oczekiwania i obrotu*

## Wyniki testowania algorytmów zastępowania stron

#### Eksperyment 1: Sekwencje stron o różnych długościach przy różnych pojemnościach pamięci

1. **Sekwencja: [**1, 2, 3, 4, 2, 2, 2, 1, 5, 6, 3**]**

|  |  |  |
| --- | --- | --- |
|  | PAGE FAULT'S | |
| POJEMNOŚĆ | FIFO | LRU |
| 3 | 8 | 8 |
| 4 | 6 | 7 |

*Tabela 3: Ilość page fault’s dla danej pojemności*

*Obraz zawierający linia, tekst, Wykres, diagram

Opis wygenerowany automatycznie*

*Wykres 3: Porównanie liczby błedów stron (1 sekwencja)*

1. **Sekwencja: [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2]**

|  |  |  |
| --- | --- | --- |
|  | PAGE FAULT'S | |
| POJEMNOŚĆ | FIFO | LRU |
| 3 | 12 | 10 |
| 4 | 9 | 7 |

*Tabela 3: Ilość page fault’s dla danej pojemności*

*Obraz zawierający linia, tekst, Wykres, diagram

Opis wygenerowany automatycznie*

*Wykres 4 Porównanie liczby błedów stron (2 sekwencja)*

#### Eksperyment 2: Losowe sekwencje stron przy różnych pojemnościach pamięci

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | PAGE FAULT'S | |
| LICZBA STRON | POJEMNOŚĆ | FIFO | LRU |
| 10 | 3 | 7 | 7 |
| 50 | 27 | 25 |
| 100 | 54 | 51 |
| 10 | 5 | 6 | 6 |
| 50 | 18 | 20 |
| 100 | 28 | 28 |

*Tabela 4: Ilość page fault’s dla danej pojemności i danej liczby stron*

*Obraz zawierający tekst, linia, diagram, Wykres

Opis wygenerowany automatycznie*

*Wykres 5: Porównanie liczby błedów stron dla różnej liczby stron i pojemności pamięci*

# Wnioski

## Alogrytmy zarządzania procesami .

 **FCFS (First-Come First-Served)**:

* W eksperymencie z procesami o losowych czasach nadejścia i stałym czasie wykonania, algorytm FCFS wykazywał krótszy średni czas oczekiwania oraz czas obrotu w porównaniu do Round Robin dla każdej liczby procesów. Szczególnie zauważalne było to przy większej liczbie procesów (125), gdzie różnica była najbardziej wyraźna.
* W przypadku procesów o stałym czasie nadejścia i losowym czasie wykonania, FCFS również okazał się bardziej efektywny, chociaż różnice były mniejsze. Algorytm ten generował mniejsze średnie czasy oczekiwania i obrotu niż Round Robin w każdej z analizowanych grup procesów (25, 75, 125) .

 **Round Robin**:

* Round Robin z ustalonym kwantem czasu (2) wypadł gorzej w porównaniu do FCFS pod względem średnich czasów oczekiwania i obrotu w obu eksperymentach. Algorytm ten lepiej radzi sobie z równoważeniem obciążenia procesów, co jest korzystne w systemach wymagających fair-share scheduling, jednakże w testach wypadł gorzej pod względem wydajności w porównaniu do FCFS .

## Alogrytmy zastępowania stron

 **FIFO (First-In-First-Out)**:

* W testach z sekwencjami stron o różnych długościach i różnych pojemnościach pamięci, algorytm FIFO wykazywał większą liczbę błędów stron w porównaniu do LRU. Szczególnie widoczne było to w 1 eksperymencie i 2 sekwencji o długości 15 stron gdzie liczba błędów stron wynosiła była o 2 mniejsza na korzyść LRU w obu przypadkach pojemności .
* W eksperymentach z losowymi sekwencjami stron FIFO również wypadał gorzej w porównaniu do LRU, zwłaszcza przy większych liczbach stron..

 **LRU (Least Recently Used)**:

* Algorytm LRU wykazywał mniejszą liczbę błędów stron w większości testowanych scenariuszy. Przy mniejszych pojemnościach pamięci, przewaga LRU nad FIFO była bardziej wyraźna, co sugeruje, że LRU lepiej radzi sobie z optymalizacją pamięci poprzez efektywniejsze zarządzanie najmniej używanymi stronami .

## Końcowe wnioski

 **Algorytmy zarządzania procesami**: FCFS wydaje się być bardziej wydajny pod względem średnich czasów oczekiwania i obrotu w porównaniu do Round Robin, co czyni go bardziej odpowiednim w systemach, gdzie czas zakończenia procesów jest kluczowy.

 **Algorytmy zastępowania stron**: LRU pokazuje lepsze wyniki w redukcji liczby błędów stron niż FIFO, szczególnie przy mniejszych pojemnościach pamięci i większych sekwencjach stron. Algorytm ten jest bardziej efektywny w systemach wymagających częstego dostępu do dużych zestawów danych .