Projekt Symulacji Algorytmów Systemy Operacyjne

Marcin Konarski 272248

Wstęp

Ten dokument jest sprawozdaniem do projektu mającego na celu symulację trzech różnych algorytmów planowania czasu procesora oraz czterech algorytmów zastępowania stron w pamięci w kontekście systemu operacyjnego. Celem eksperymentu jest analiza i porównanie działania oraz efektywności tych algorytmów, a także zrozumienie ich wpływu na wydajność systemu operacyjnego w kontekście czasu wykonania procesów oraz zarządzania pamięcią. Implementacja tych elementów pozwala na szersze zrozumienie mechanizmów działania systemu operacyjnego pod kątem planowania zadań i zarządzania zasobami. W pierwszej kolejności każdy algorytm zostanie omówiony, zostanie przedstawiona jego implementacja w programie oraz zostanie krótko opisane jego działanie. Po przedstawieniu algorytmów i prezentacji wyników testów, nastąpi ich porównanie oraz wyciągnięcie wniosków. Oto rozpatrywane algorytmy:

- First Come First Serve CPU Time Scheduling Algorithm
- First Come First Serve CPU Time Scheduling Algorithm With Aging
- Shortest Job First CPU Time Scheduling Algorithm
- First In First Out Page Replacement Algorithm
- Optimal Page Replacement Algorithm
- Least Recently Used Page Replacement Algorithm
- Least Frequently Used Page Replacement Algorithm

Sposób testowania obejmuje wprowadzenie różnych, losowych, spełniających założenia prowadzącego próbek danych testowych do pliku, odpalenie programu main.exe i wybór odpowiedniego algorytmu bądź raportów generowanych przez program oraz analizę i porównanie danych. Testy obejmowały różne scenariusze aby zweryfikować poprawność działania algorytmów oraz dokonać ich oceny i porównania.

Symulacje Algorytmów Planowania Czasu Procesora

Implementacja FCFS

FCFS (First-Come-First-Served) to algorytm planowania czasu procesora, który przydziela procesy do wykonania w kolejności, w jakiej nadeszły do systemu operacyjnego. Oznacza to, że pierwszy proces, który przybył, zostanie obsłużony jako pierwszy. Jest to prosty, intuicyjny i łatwy w implementacji algorytm, jednakże może prowadzić do tzw. "convoy effect", jest to sytuacja a której długi proces, który zajmuje więcej czasu na wykonanie, blokuje dostęp do zasobów procesom krótszym.

```
class FCFS:
  def __init__(self, id, arrival_time, execution_time, turnaround_time, waiting_time, time_until_finish):
    self.id = id
    self.arrival_time = arrival_time
    self.execution_time = execution_time
    self.turnaround_time = turnaround_time
    self.waiting_time = waiting_time
    self.time_until_finish = time_until_finish
    self.average_turnaround_time = 0
    self.average_waiting_time = 0

  def fcfs_simulation(self, processes):
    execution_list = [i.execution_time for i in processes]
    processes.sort(key=lambda x: x.arrival_time)
    current_time = 0
    for process in processes:
      process.waiting_time = max(0, current_time - process.arrival_time)
      process.turnaround_time = current_time + process.execution_time - process.arrival_time
      process.time_until_finish = current_time + process.execution_time
      current_time = process.time_until_finish
    self.average_turnaround_time = sum(process.turnaround_time for process in processes) / len(processes)
    self.average_waiting_time = sum(process.waiting_time for process in processes) / len(processes)
```

Klasa FCFS implementuje algorytm planowania procesów. W metodzie fcfs_simulation, procesy są sortowane rosnąco według czasu ich przyjścia. Następnie, dla każdego procesu, obliczane są parametry takie jak czas oczekiwania (waiting_time), czas trwania (turnaround_time) oraz czas do zakończenia (time_until_finish) na podstawie aktualnego czasu. Algorytm iteruje przez posortowane procesy, aktualizując parametry każdego z nich. Po zakończeniu iteracji, obliczane są średnie czasy trwania i oczekiwania dla wszystkich procesów.

Implementacja SJF

SJF (Shortest Job First) to algorytm planowania czasu procesora, który przydziela procesy do wykonania według długości ich czasu trwania. Z pośród procesów, oczekujących proces o najkrótszym czasie trwania zostanie obsłużony jako pierwszy. W przypadku algorytmu SJF, procesy są sortowane rosnąco według czasu trwania, a następnie są wykonywane zgodnie z tą kolejnością. SJF ma potencjał zminimalizowania czasu oczekiwania, jednakże może prowadzić do zjawiska "niesprawiedliwości" dla dłuższych procesów.

```
  def sjf_simulation(self, processes):
    execution_list = [i.execution_time for i in processes]
    processes_done = []
    processes.sort(key=lambda x: x.arrival_time)
    current_time = 0
    queue = processes.copy()

    while queue:
      ready_processes = [process for process in queue if process.arrival_time <= current_time]
      if ready_processes:
        shortest_job = min(ready_processes, key=lambda x: x.execution_time)
        queue.remove(shortest_job)

        shortest_job.waiting_time = current_time - shortest_job.arrival_time
        self.waiting_time += shortest_job.waiting_time
        current_time += shortest_job.execution_time
        shortest_job.time_until_finish = current_time
        shortest_job.turnaround_time = current_time - shortest_job.arrival_time
        processes_done.append(shortest_job)
      else:
        current_time += 1
    self.average_waiting_time = sum([process.waiting_time for process in processes]) / len(processes)
    self.average_turnaround_time = sum([process.turnaround_time for process in processes]) / len(processes)
```

Klasa SJF implementuje algorytm planowania procesów. Metoda sjf_simulation sortuje procesy według czasu ich przyjścia, a następnie iteruje przez nie, wybierając proces o najkrótszym czasie trwania spośród gotowych do wykonania. Dla każdego wybranego procesu obliczane są czasy oczekiwania, czasu trwania oraz czasu do zakończenia. Po zakończeniu iteracji obliczane są średnie czasy trwania i oczekiwania dla wszystkich procesów.

Implementacja FCFS z wykorzystaniem postarzania procesów

FCFS with Aging to modyfikacja standardowego algorytmu FCFS, która wprowadza element postarzania procesów. Każdemu procesowi przypisywany jest priorytet, który zmniejsza się w czasie oczekiwania na wykonanie. Im dłużej proces czeka, tym niższy ma priorytet. Procesy są obsługiwane zgodnie z ich priorytetem, a element postarzania pomaga uniknąć sytuacji, w której długi proces blokuje dostęp do zasobów. Element postarzania pomaga zminimalizować ryzyko zjawiska "convoy effect" - wspomnianego wcześniej przy opisie standardowego algorytmu FCFS - oraz utrzymanie równowagi w dostępie do zasobów systemu.

```
class FCFS_with_aging:
  def __init__(self, id, arrival_time, execution_time, turnaround_time, waiting_time, time_until_finish, priority):
    self.id = id
    self.arrival_time = arrival_time
    self.execution_time = execution_time
    self.turnaround_time = turnaround_time
    self.waiting_time = waiting_time
    self.time_until_finish = time_until_finish
    self.priority = priority
    self.average_turnaround_time = 0
    self.average_waiting_time = 0

  def aging_implementaton(self, queue, current_time, quantum_variable):
    ready_processes_list = [process for process in queue if process.arrival_time <= current_time]
    for process in ready_processes_list:
      process.waiting_time = current_time - process.arrival_time
      if process.waiting_time >= quantum_variable:
        process.priority -= (process.waiting_time // quantum_variable)
        process.priority = max(0, process.priority)
    return ready_processes_list

  def fcfs_with_aging_simulation(self, processes, quantum_variable):
    execution_list = [process.execution_time for process in processes]
    processes_done = []
    current_time = 0
    queue = processes.copy()
    while queue:
      ready_processes = self.aging_implementaton(queue, current_time, quantum_variable)
      if ready_processes:
        current_process = min(ready_processes, key=lambda x: x.priority)
        queue.remove(current_process)
        current_process.waiting_time = current_time - current_process.arrival_time
        current_time += current_process.execution_time
        current_process.time_until_finish = current_time
        current_process.turnaround_time = current_time - current_process.arrival_time
        processes_done.append(current_process)
      else:
        current_time += 1
    self.average_waiting_time = sum([process.waiting_time for process in processes]) / len(processes)
    self.average_turnaround_time = sum([process.turnaround_time for process in processes]) / len(processes)
```

W metodzie fcfs_with_aging_simulation, procesy są inicjalizowane zgodnie z kolejnością przyjścia. W głównej pętli, procesy są obsługiwane zgodnie z ich priorytetem, który uwzględnia element postarzania. Procesy są usuwane z kolejki w momencie rozpoczęcia ich wykonania. Dla każdego procesu obliczane są czas oczekiwania, czas trwania i czas do zakończenia. Po zakończeniu iteracji, obliczane są średnie czasy trwania i oczekiwania dla wszystkich procesów

W metodzie aging_implementation, procesy, których czas przybycia jest mniejszy niż "obecny czas" (current_time) dodawane do listy ready_processes co pomaga zasymulować obsługę procesów "w czasie rzeczywistym". Następnie są one sortowane względem najpierw priorytetu i następnie czasu przyjścia. W pętli dla każdego z gotowych procesów, obliczane są czasy oczekiwania z uwzględnieniem postarzania. Procesy, które czekały wystarczająco długo, są zmniejszane zgodnie z określonym kwantem (quantum_variable). Maksymalny priorytet wynosi 0.

Symulacje Algorytmów Zastępowania Stron

Implementacja FIFO

Algorytm FIFO (First-In-First-Out) jest prostym algorytmem zastępowania stron, który bazuje na koncepcji kolejki. Zakłada, że strony, które znajdują się w pamięci, są traktowane w kolejności, w jakiej zostały do niej wprowadzone. Gdy dochodzi do błędu strony (page fault), czyli danej strony brak w pamięci, algorytm FIFO usuwa z pamięci tę stronę, która była w niej najdłużej, czyli tę, która znajduje się na początku kolejki. Następnie nowa strona jest dodawana na koniec kolejki, symbolizując, że jest ona teraz najnowszą stroną w pamięci. Algorytm ten jest łatwy do zrozumienia i implementacji, jednak nie zawsze zapewnia optymalne wyniki, szczególnie w przypadku pewnych wzorców dostępu do pamięci.

```
class FIFO:
  def __init__(self, frames_num):
    self.frames_num = frames_num
    self.page_frames = []
    self.faults = 0
    self.faults_ratio = 0
    self.visualization_data = []

  def fifo_simulation(self, pages_list):
      index = 0
      for page in pages_list:
        if page in self.page_frames:
          self.visualization_data.append(None)
        else:
          if len(self.page_frames) < self.frames_num:
            self.page_frames.append(page)
            self.faults += 1
            self.visualization_data.append((self.page_frames.copy(), self.faults))
          else:
            if len(self.page_frames) < self.frames_num:
              self.page_frames.append(page)
            else:
              self.page_frames[index] = page
            if index < self.frames_num - 1:
              index += 1
            else:
              index = 0
            self.faults += 1
            self.visualization_data.append((self.page_frames.copy(), self.faults))
      self.faults_ratio = self.faults / len(pages_list)
```

Metoda fifo_simulation ma na celu iterację po wszystkich stronach z ciągu odniesień (pobieranego z pliku). Jeżeli strona nie znajduje się w ramce wtedy oznacza page_fault i jeżeli jest jakaś pusta ramka to dodaje do niej stronę, w przeciwnym wypadku usuwana jest pierwsza strona z ramki i dodawana jest ocena na sam koniec spełniając założenie "first in first out". Jeżeli jednak strona znajduje się już w ramce następuje page hit co jest oznaczane jako dodanie do listy, mającej na celu wyświetlanie całego procesu, wartości None.

Implementacja OPT

Algorytm OPT (Optimal) page replacement, jest teoretycznym, optymalnym algorytmem zastępowania stron, który służy jako punkt odniesienia dla porównań z innymi algorytmami. W przeciwieństwie do wielu praktycznych algorytmów zastępowania stron, OPT zakłada pełną wiedzę o przyszłych odwołaniach do stron. Algorytm ten zawsze zastępuje tę stronę, która nie będzie używana najdłuższy czas. W praktyce, ze względu na brak wiedzy o przyszłych stronach, implementacja rzeczywista tego algorytmu jest niemożliwa. Mimo to, OPT stanowi dobry punkt odniesienia, pozwalając na ocenę wydajności innych algorytmów zastępowania stron w porównaniu do optymalnego przypadku.

```
class OPT:
  def __init__(self, frames_num):
    self.frames_num = frames_num
    self.page_frames = []
    self.faults = 0
    self.visualization_data = []
    self.fault_ratio = 0

  def opt_simulation(self, pages_list):
    for i, page in enumerate(pages_list):
      found = any(frame == page for frame in self.page_frames)
      if found:
        self.visualization_data.append(None)
        continue

      if len(self.page_frames) < self.frames_num:
        self.page_frames.append(page)
        self.faults += 1
      else:
        remaining_pages = pages_list[i + 1:]
        frame_distances = {}
        for frame in self.page_frames:
          if frame in remaining_pages:
            frame_distances[frame] = remaining_pages.index(frame)
          else:
            frame_distances[frame] = float('inf')
        page_to_replace = max(frame_distances, key=frame_distances.get)
        replaceIndex = self.page_frames.index(page_to_replace)
        self.page_frames[replaceIndex] = page
        self.faults += 1

      self.visualization_data.append((self.page_frames.copy(), self.faults
    self.fault_ratio = self.faults / len(pages_list)
```

W metodzie opt_simulation dla każdej strony, jeżeli obecnie rozpatrywana strona znajduje się w ramce to do do listy, mającej na celu wyświetlanie całego procesu zostaje dodane wartości None symulująca brak zmiany strony. Jeżeli jest jakaś pusta ramka to dodaje do niej stronę, w przeciwnym wypadku tworzy listę z przyszłymi stronami od momentu obecnie rozpatrywanej strony, przyporządkowuje każdej stronie z obecnie znajdujących się w ramce jej odległość z listy przyszłych stron, jeżeli strony nie ma na liście to przypisuje jej wartość ‘inf’ i zastępuje stronę o największej odlegości, czyli tę która najpóźniej będzie użyta.

Implementacja LRU

Algorytm LRU (Least Recently Used) jest stosunkowo prosty do zrozumienia i implementacji, jest również popularnym wyborem w systemach operacyjnych i bazach danych, gdzie efektywne zarządzanie pamięcią jest kluczowe dla wydajności systemu. Algorytm bazuje na zasadzie, że strony, które były najmniej używane, są najbardziej prawdopodobne do zastąpienia. W kontekście zarządzania pamięcią, każda strona ma przypisaną informację o tym, kiedy była ostatnio używana. Algorytm utrzymuje listę, zwyczajowo zwaną "access order" lub "order of access", w której zapisywane są strony w kolejności ich ostatniego użycia, gdzie najnowsza strona znajduje się na początku listy, a najstarsza na końcu. Gdy nowa strona musi zostać wprowadzona do pamięci, a miejsce w ramkach pamięci jest ograniczone, algorytm LRU identyfikuje i usuwa ze zbioru tę stronę, która nie była używana przez najdłuższy czas, co można określić na podstawie końca listy "access order". Następnie nowa strona jest dodawana na początek listy, sygnalizując, że jest teraz najbardziej aktualnie używaną.

```python
class LRU:
    def __init__(self, frames_num):
        self.frames_num = frames_num
        self.page_frames = []
        self.faults = 0
        self.visualization_data = []
        self.access_order = []  ## a list to maintain the order of page accesses
        self.fault_ratio = 0

    def lru_simulation(self, pages_list):
        for page in pages_list:
            if page in self.page_frames:
                # Page is already in frames, move it to the front to indicate most recent usage
                self.access_order.remove(page)
                self.access_order.insert(0, page)
                self.visualization_data.append(None)
            else:
                if len(self.page_frames) < self.frames_num:
                    self.page_frames.append(page)
                    self.access_order.insert(0, page)
                    self.visualization_data.append((self.page_frames.copy(), self.faults))
                else:
                    # If there are no empty frames, remove the least recently used page and append the new one
                    lru_page = self.access_order.pop()
                    index = self.page_frames.index(lru_page)
                    self.page_frames[index] = page
                    self.access_order.insert(0, page)
                    self.visualization_data.append((self.page_frames.copy(), self.faults))

                self.faults += 1

        self.fault_ratio = self.faults / len(pages_list)
```

Algorytm iteruje zażdą stronę w pages_list, czyli ciągu stron pobieranym z pliku. Jeżeli znajdzie stronę w ramce to ustawia ją na sam początek listy access_order, w celu wskazania, że jest tą Recently Used. Jeżeli w ramce jest jednak puste miejsce to dodaje do ramki tę stronę oraz aktualizuje access_order. Jeżeli jednak nie znajdzie w ramce danej strony wówczas dodaje ją do ramki i o listy access_order oraz usuwa zarówno z access_order jak i z ramki tę stronę, która była najdawniej wykorzystywana.

Implementacja LFU

Algorytm LFU (Least Frequently Used) działa na zasadzie, że strony, które były używane najrzadziej, są najbardziej prawdopodobne do zastąpienia. W przypadku LFU, każda strona w pamięci ma przypisaną liczbę określającą, jak często była używana. Algorytm monitoruje częstotliwość dostępu do poszczególnych stron i decyduje o zastępowaniu strony o najniższej częstotliwości, gdy dochodzi do błędu strony. W momencie, gdy nowa strona musi być wprowadzona do pamięci, a ilość dostępnych ramek pamięci jest ograniczona, LFU wybiera stronę, która miała najmniejszą ilość odwołań.

```python
class LFU:
    def __init__(self, frames_num):
        self.frames_num = frames_num
        self.page_frames = []
        self.faults = 0
        self.visualization_data = []
        self.page_frequency = {}
        self.fault_ratio = 0

    def lfu_simulation(self, pages_list):
        for page in pages_list:
            if page in self.page_frames:
                self.page_frequency[page] += 1
                self.visualization_data.append(None)
            else:
                self.faults += 1
                if len(self.page_frames) < self.frames_num:
                    self.page_frames.append(page)
                    self.page_frequency[page] = 1
                else:
                    min_page = min(self.page_frequency, key=self.page_frequency.get)
                    self.page_frames[self.page_frames.index(min_page)] = page
                    self.page_frequency.pop(min_page)
                    self.page_frequency[page] = 1

                self.visualization_data.append((self.page_frames.copy(), self.faults))
        self.fault_ratio = self.faults / len(pages_list)
```

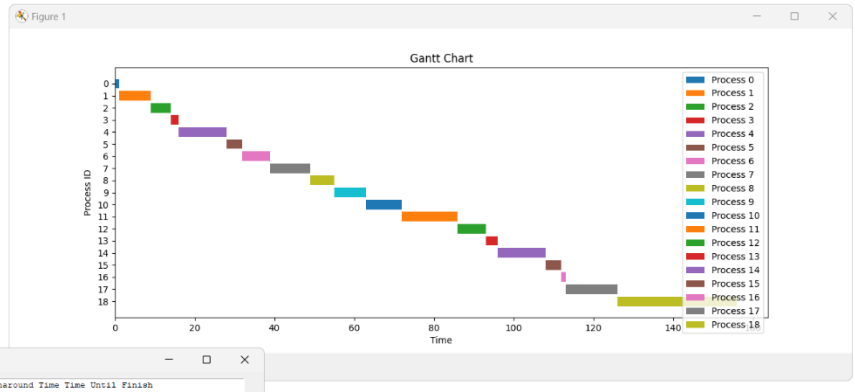
Algorytm, dla każdej strony, jeżeli znajdzie ją w ramce wtedy zwiększa częstotliwość danej strony i symuluje page hit. Powiązanie stron wraz z ich częstotliwościami jest uzyskane za pomocą dictionary o nazwie page_frequency. Jeżeli strony nie ma w ramce wówczas ocznacza page fault, szuka strony o najmniejszej częstotliwości wystąpień, usuwa ją z page_frequency, zastępuje ją nową stroną oraz ustawia ilość wystąpień nowej strony na jeden.

Porównanie wyników dla
Algorytmów Planowania Czasu Procesora

```
0 0 1 2 5 7 7 9 13 14 15 18 21 28 31 42 54 68
1 8 5 2 12 4 7 10 6 8 9 14 7 3 12 4 1 13 30 7
5 12 18 20 15 3 10 9 2 6 8 16 4 19 11 7 14 0 5
20
4
```

Obraz przedstawia przykładowe dane wejściowe wprowadzane z pliku w formacie ustalonym arbitralnie, gdzie pierwsza linijka oznacza czas przybycia procesów, druga czas wykonywania procesów a trzecia priorytety dla każdego z procesów. Czwarta i piątka linijka pliku wejściowego oznaczają kojeno: kwant czasu dla alg priorytetowo, w tym przypadku jest to fcfs z wykorzystaniem mechaniki postarzania procesów, oraz kwant czasu dla alg RoundRobin, któregy ostatecznie nie jest wykorzystywany w mojej realizacji projektu. Oto wyniki testów dla kolejnych danych dla algorytmów:

FCFS

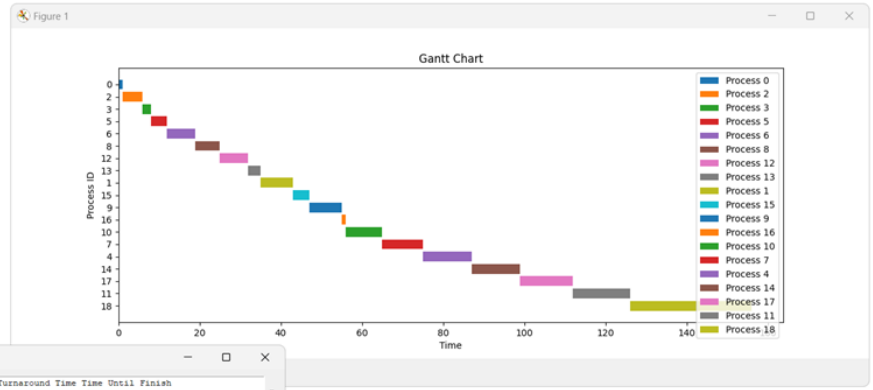


FCFS Simulation

ID	Arrival Time	Execution Time	Waiting Time	Turnaround Time	Time Until Finish
0	0	1	0	1	1
1	0	8	1	9	9
2	1	5	8	13	14
3	2	2	12	14	16
4	5	12	11	23	28
5	7	4	21	25	32
6	7	7	25	32	39
7	9	10	30	40	49
8	13	6	36	42	55
9	14	8	41	49	63
10	15	9	48	57	72
11	18	14	54	68	86
12	21	7	65	72	93
13	28	3	65	68	96
14	31	12	65	77	108
15	42	4	66	70	112
16	54	1	58	59	113
17	68	13	45	58	126
18	80	30	46	76	156

```
Total time for all processes to finish: 156
Average turnaround time: 44.89473684210526
Average waiting time: 36.68421052631579
```

SJF

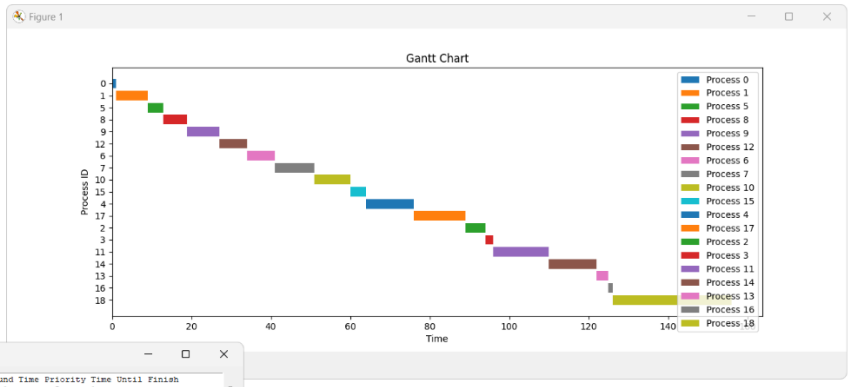


SJF Simulation

ID	Arrival Time	Execution Time	Waiting Time	Turnaround Time	Time Until Finish
0	0	1	0	1	1
2	1	5	0	5	6
3	2	2	4	6	8
5	7	4	1	5	12
6	7	7	5	12	19
8	13	6	6	12	25
12	21	7	4	11	32
13	28	3	4	7	35
1	0	8	35	43	43
15	42	4	1	5	47
9	14	8	33	41	55
16	54	1	1	2	56
10	15	9	41	50	65
7	9	10	56	66	75
4	5	12	70	82	87
14	31	12	56	68	99
17	68	13	31	44	112
11	18	14	94	108	126
18	80	30	46	76	156

```
Total time for all processes to finish: 156
Average turnaround time: 33.89473684210526
Average waiting time: 25.68421052631579
```

FCFS z wykorzystaniem postarzania procesów



FCFS With Aging Simulation

ID	Arrival Time	Execution Time	Waiting Time	Turnaround Time	Priority	Time Until Finish
0	0	1	0	1	5	1
1	0	8	1	9	12	9
5	7	4	2	6	3	13
8	13	6	0	6	2	19
9	14	8	5	13	6	27
12	21	7	6	13	4	34
6	7	7	27	34	8	41
7	9	10	32	42	7	51
10	15	9	36	45	6	60
15	42	4	18	22	7	64
4	5	12	59	71	6	76
17	68	13	8	21	0	89
2	1	5	88	93	0	94
3	2	2	92	94	0	96
11	18	14	78	92	0	110
14	31	12	79	91	0	122
13	28	3	94	97	0	125
16	54	1	71	72	0	126
18	80	30	46	76	0	156

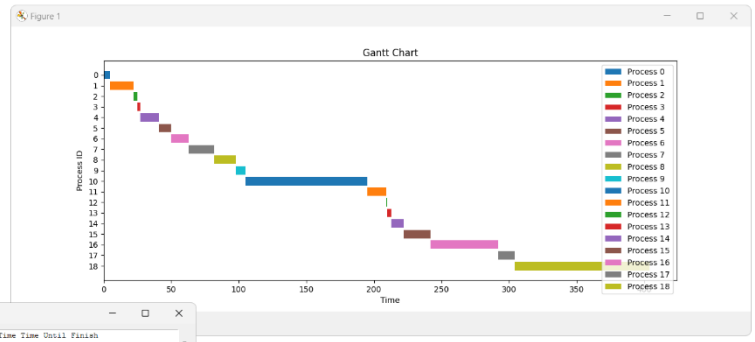
```
Total time for all processes to finish: 156
Average turnaround time: 47.26315789473684
Average waiting time: 39.05263157894737
```

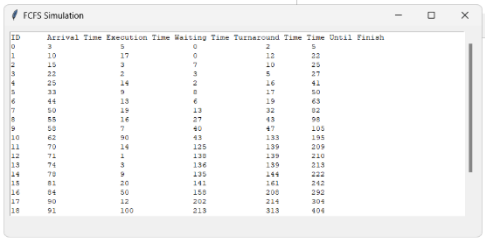
Porównanie wyników dla

Algorytmów Planowania Czasu Procesora Dla Innego Zestawu Danych

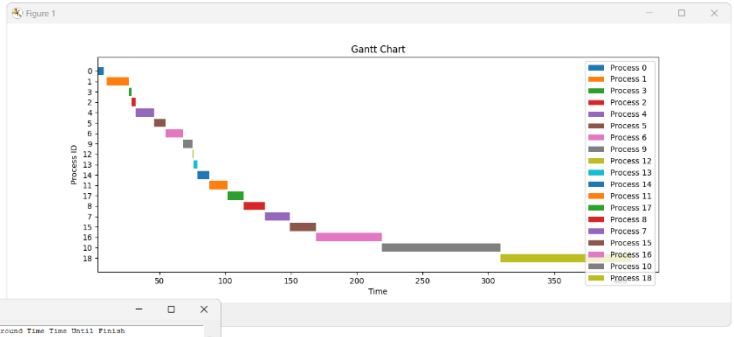
```
3 10 15 22 25 33 44 50 55 58 62 70 71 74 78 81 84 90 91
5 17 3 2 14 9 13 19 16 7 90 14 1 3 9 20 50 12 100
12 4 8 15 1 7 14 18 11 16 3 90 66 23 10 130 17 119 40
31
5
```

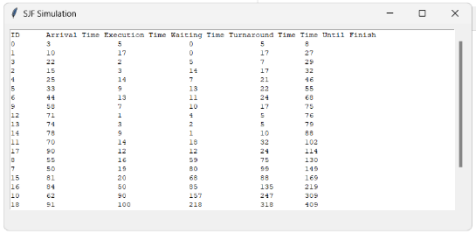
FCFS



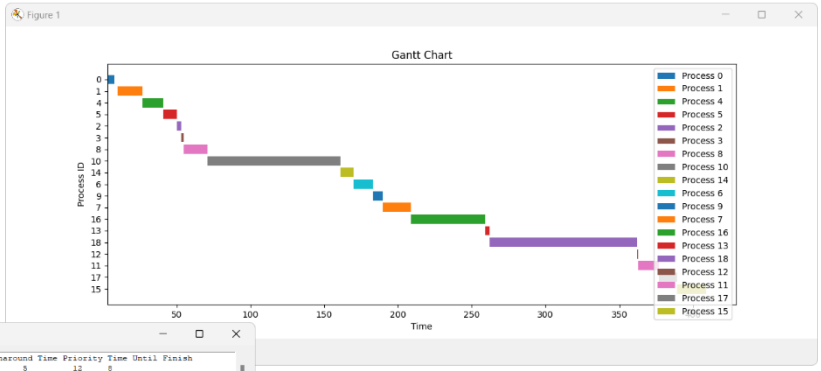


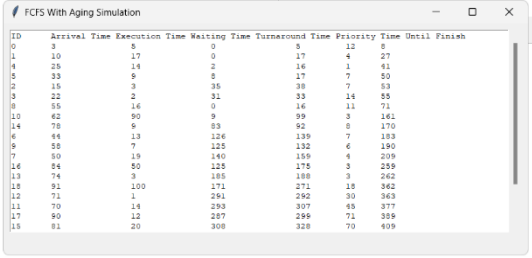
SJF





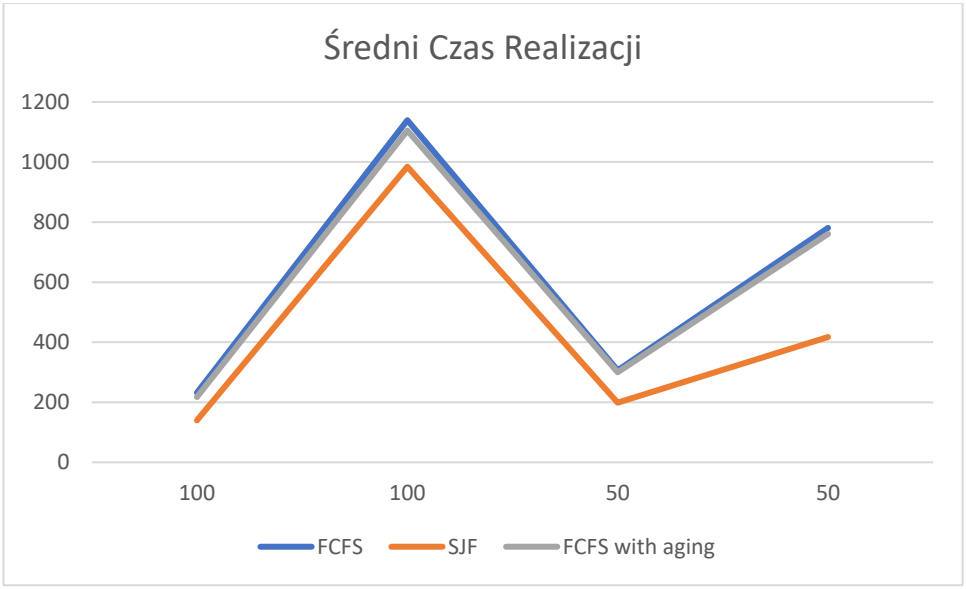
FCFS z wykorzystaniem postarzania procesów

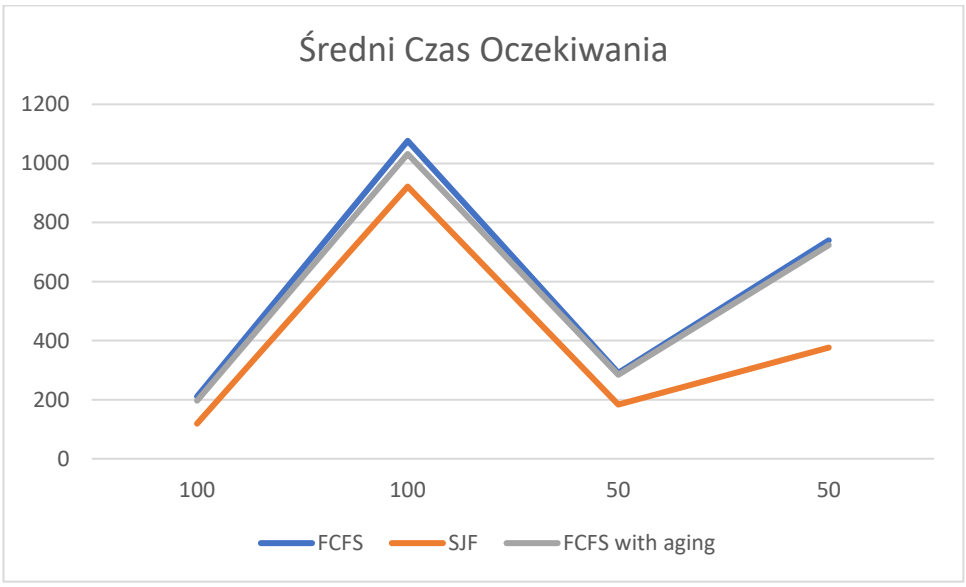




Porównanie Algorytmów

Porównanie Algorytmów ze względu na średni czas realizacji oraz średni czas oczekiwania porcesów w zależności od wielkości losowej próbki (oś X reprezentuje ilość procesów z losowo dobranymi danymi branych pod uwagę, oś Y reprezentuje czas – libczę taktów)





Wyniki są bardzo podobne niezależnie od badania średniego czasu realizacji czy średniego czasu oczekiwania procesów. FCFS osiąga najgorsze wyniki zajmując najwięcej czasu. FCFS z zaimplementowanym mechanizmem postarzania w przypadku procesów z priorytetami osiąga nieznacznie lepsze wyniki, jednakże różnica między tymi dwoma algrytmami może się różnić w zależności od warunków i wartości procesów jaki przychodzą do CPU. Bez zaskoczenia jest fakt, że SJF osiąga najlepsze wyniki – podczas korzystania z tego algorytmu czasy zarówno oczekiwania jak i realizacji procesów są najlepsze.

Wnioski Dla Algorytmów Planowania Czasu Procesora

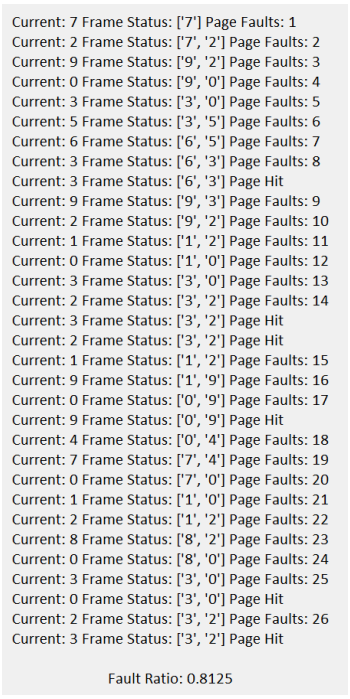
Wszystkie algorytmy zachowują się zgodnia z przewidywaniami. FCFS ma zazwyczas najgorsze wyniki średnich czasów czekania i wykonywania, FCFS z wykorzystaniem postarzania procesów pomaga nieznacznie poprawić te wyniki, jednakże jak widać na pierwszym zdjęciu jeżeli przyjdzie proces o długim czasie wykonywania, lecz bardzo małym priorytecie wtedy średni czas wykonywania dla wszystkich procesó znacząco się zwiększa. Z pośród tych trzech algorytmów, dla testowanych danych SJF jest tym optymalnym algorytmem, ponieważ osiąga najkrówsze średnie czasy. Algorytm FCFS jest prostym algorytmem, ale może prowadzić do niesprawiedliwości w obsłudze krótkich procesów. SJF minimalizuje czas oczekiwania, ale wymaga informacji o czasach wykonywania. Algorytm FCFS z postarzaniem procesów ma potencjał minimalizowania efektu convoy, ale również wymaga przewidywania czasów wykonywania. Każdy algorytm ma swoje zalety i wady, a wybór zależy od specyfiki danego systemu oraz rodzaju procesów, jakie musi obsługiwać.

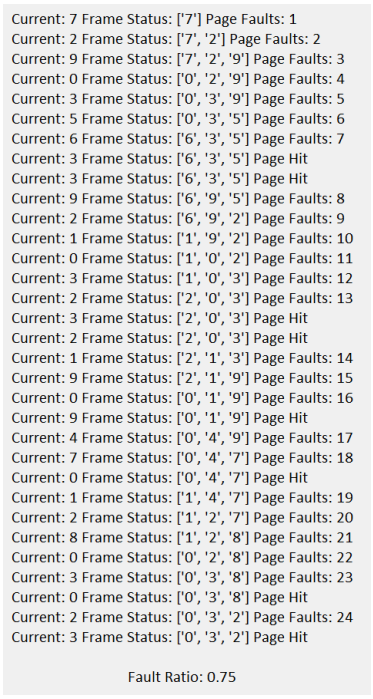
Porównanie wyników dla Algorytmów Zastępowania Stron

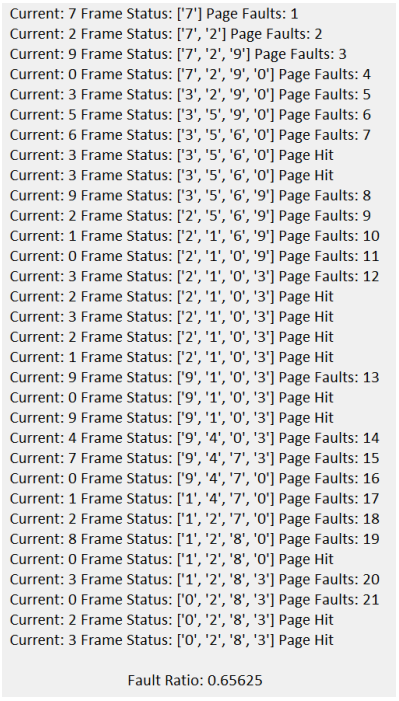
Poniżej przedstawiono przykładowe dane wejściowe, które są wczytywane z pliku, przy czym format danych został ustalony arbitralnie – jest to ciąg odniesień wraz z numerami stron liczba ramek została zmieniana na bieżąco. Założeniem zadania jest fakt, że liczba ramek nie może być większa niż 5 zatem po wprowadzeniu błędnych danych wyskakuje odpowiedni komunikat. Poniżej przedstawiono wyniku testów dla różnej ilości ramek dla wszystkich algorytmów.

7 2 9 0 3 5 6 3 3 9 2 1 0 3 2 3 2 1 9 0 9 4 7 0 1 2 8 0 3 0 2 3

FIFO dla różnej ilości ramek:







OPT dla różnej ilości ramek:

Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['9', '2'] Page Faults: 3
Current: 0 Frame Status: ['9', '0'] Page Faults: 4
Current: 3 Frame Status: ['9', '3'] Page Faults: 5
Current: 5 Frame Status: ['5', '3'] Page Faults: 6
Current: 6 Frame Status: ['6', '3'] Page Faults: 7
Current: 3 Frame Status: ['6', '3'] Page Hit
Current: 3 Frame Status: ['6', '3'] Page Hit
Current: 9 Frame Status: ['9', '3'] Page Faults: 8
Current: 2 Frame Status: ['2', '3'] Page Faults: 9
Current: 1 Frame Status: ['1', '3'] Page Faults: 10
Current: 0 Frame Status: ['0', '3'] Page Faults: 11
Current: 3 Frame Status: ['0', '3'] Page Hit
Current: 2 Frame Status: ['2', '3'] Page Faults: 12
Current: 3 Frame Status: ['2', '3'] Page Hit
Current: 2 Frame Status: ['2', '3'] Page Hit
Current: 1 Frame Status: ['2', '1'] Page Faults: 13
Current: 9 Frame Status: ['9', '1'] Page Faults: 14
Current: 0 Frame Status: ['9', '0'] Page Faults: 15
Current: 9 Frame Status: ['9', '0'] Page Hit
Current: 4 Frame Status: ['4', '0'] Page Faults: 16
Current: 7 Frame Status: ['7', '0'] Page Faults: 17
Current: 0 Frame Status: ['7', '0'] Page Hit
Current: 1 Frame Status: ['1', '0'] Page Faults: 18
Current: 2 Frame Status: ['2', '0'] Page Faults: 19
Current: 8 Frame Status: ['8', '0'] Page Faults: 20
Current: 0 Frame Status: ['8', '0'] Page Hit
Current: 3 Frame Status: ['3', '0'] Page Faults: 21
Current: 0 Frame Status: ['3', '0'] Page Hit
Current: 2 Frame Status: ['3', '2'] Page Faults: 22
Current: 3 Frame Status: ['3', '2'] Page Hit

Fault Ratio: 0.6875

Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['7', '2', '9'] Page Faults: 3
Current: 0 Frame Status: ['0', '2', '9'] Page Faults: 4
Current: 3 Frame Status: ['3', '2', '9'] Page Faults: 5
Current: 5 Frame Status: ['3', '5', '9'] Page Faults: 6
Current: 6 Frame Status: ['3', '6', '9'] Page Faults: 7
Current: 3 Frame Status: ['3', '6', '9'] Page Hit
Current: 3 Frame Status: ['3', '6', '9'] Page Hit
Current: 9 Frame Status: ['3', '6', '9'] Page Hit
Current: 2 Frame Status: ['3', '2', '9'] Page Faults: 8
Current: 1 Frame Status: ['3', '2', '1'] Page Faults: 9
Current: 0 Frame Status: ['3', '2', '0'] Page Faults: 10
Current: 3 Frame Status: ['3', '2', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '0'] Page Hit
Current: 3 Frame Status: ['3', '2', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '0'] Page Hit
Current: 1 Frame Status: ['1', '2', '0'] Page Faults: 11
Current: 9 Frame Status: ['1', '9', '0'] Page Faults: 12
Current: 0 Frame Status: ['1', '9', '0'] Page Hit
Current: 9 Frame Status: ['1', '9', '0'] Page Hit
Current: 4 Frame Status: ['1', '4', '0'] Page Faults: 13
Current: 7 Frame Status: ['1', '7', '0'] Page Faults: 14
Current: 0 Frame Status: ['1', '7', '0'] Page Hit
Current: 1 Frame Status: ['1', '7', '0'] Page Hit
Current: 2 Frame Status: ['2', '7', '0'] Page Faults: 15
Current: 8 Frame Status: ['2', '8', '0'] Page Faults: 16
Current: 0 Frame Status: ['2', '8', '0'] Page Hit
Current: 3 Frame Status: ['2', '3', '0'] Page Faults: 17
Current: 0 Frame Status: ['2', '3', '0'] Page Hit
Current: 2 Frame Status: ['2', '3', '0'] Page Hit
Current: 3 Frame Status: ['2', '3', '0'] Page Hit

Fault Ratio: 0.53125

Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['7', '2', '9'] Page Faults: 3
Current: 0 Frame Status: ['7', '2', '9', '0'] Page Faults: 4
Current: 3 Frame Status: ['3', '2', '9', '0'] Page Faults: 5
Current: 5 Frame Status: ['3', '2', '9', '5'] Page Faults: 6
Current: 6 Frame Status: ['3', '2', '9', '6'] Page Faults: 7
Current: 3 Frame Status: ['3', '2', '9', '6'] Page Hit
Current: 3 Frame Status: ['3', '2', '9', '6'] Page Hit
Current: 9 Frame Status: ['3', '2', '9', '6'] Page Hit
Current: 2 Frame Status: ['3', '2', '9', '6'] Page Hit
Current: 1 Frame Status: ['3', '2', '9', '1'] Page Faults: 8
Current: 0 Frame Status: ['3', '2', '0', '1'] Page Faults: 9
Current: 3 Frame Status: ['3', '2', '0', '1'] Page Hit
Current: 2 Frame Status: ['3', '2', '0', '1'] Page Hit
Current: 3 Frame Status: ['3', '2', '0', '1'] Page Hit
Current: 2 Frame Status: ['3', '2', '0', '1'] Page Hit
Current: 1 Frame Status: ['3', '2', '0', '1'] Page Hit
Current: 9 Frame Status: ['9', '2', '0', '1'] Page Faults: 10
Current: 0 Frame Status: ['9', '2', '0', '1'] Page Hit
Current: 9 Frame Status: ['9', '2', '0', '1'] Page Hit
Current: 4 Frame Status: ['4', '2', '0', '1'] Page Faults: 11
Current: 7 Frame Status: ['7', '2', '0', '1'] Page Faults: 12
Current: 0 Frame Status: ['7', '2', '0', '1'] Page Hit
Current: 1 Frame Status: ['7', '2', '0', '1'] Page Hit
Current: 2 Frame Status: ['7', '2', '0', '1'] Page Hit
Current: 8 Frame Status: ['8', '2', '0', '1'] Page Faults: 13
Current: 0 Frame Status: ['8', '2', '0', '1'] Page Hit
Current: 3 Frame Status: ['3', '2', '0', '1'] Page Faults: 14
Current: 0 Frame Status: ['3', '2', '0', '1'] Page Hit
Current: 2 Frame Status: ['3', '2', '0', '1'] Page Hit
Current: 3 Frame Status: ['3', '2', '0', '1'] Page Hit

Fault Ratio: 0.4375

LRU dla różnej ilości ramek:

Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['9', '2'] Page Faults: 3
Current: 0 Frame Status: ['9', '0'] Page Faults: 4
Current: 3 Frame Status: ['3', '0'] Page Faults: 5
Current: 5 Frame Status: ['3', '5'] Page Faults: 6
Current: 6 Frame Status: ['6', '5'] Page Faults: 7
Current: 3 Frame Status: ['6', '3'] Page Faults: 8
Current: 3 Frame Status: ['6', '3'] Page Hit
Current: 9 Frame Status: ['9', '3'] Page Faults: 9
Current: 2 Frame Status: ['9', '2'] Page Faults: 10
Current: 1 Frame Status: ['1', '2'] Page Faults: 11
Current: 0 Frame Status: ['1', '0'] Page Faults: 12
Current: 3 Frame Status: ['3', '0'] Page Faults: 13
Current: 2 Frame Status: ['3', '2'] Page Faults: 14
Current: 3 Frame Status: ['3', '2'] Page Hit
Current: 2 Frame Status: ['3', '2'] Page Hit
Current: 1 Frame Status: ['1', '2'] Page Faults: 15
Current: 9 Frame Status: ['1', '9'] Page Faults: 16
Current: 0 Frame Status: ['0', '9'] Page Faults: 17
Current: 9 Frame Status: ['0', '9'] Page Hit
Current: 4 Frame Status: ['4', '9'] Page Faults: 18
Current: 7 Frame Status: ['4', '7'] Page Faults: 19
Current: 0 Frame Status: ['0', '7'] Page Faults: 20
Current: 1 Frame Status: ['0', '1'] Page Faults: 21
Current: 2 Frame Status: ['2', '1'] Page Faults: 22
Current: 8 Frame Status: ['2', '8'] Page Faults: 23
Current: 0 Frame Status: ['0', '8'] Page Faults: 24
Current: 3 Frame Status: ['0', '3'] Page Faults: 25
Current: 0 Frame Status: ['0', '3'] Page Hit
Current: 2 Frame Status: ['0', '2'] Page Faults: 26
Current: 3 Frame Status: ['3', '2'] Page Faults: 27

Fault Ratio: 0.84375

Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['7', '2', '9'] Page Faults: 3
Current: 0 Frame Status: ['0', '2', '9'] Page Faults: 4
Current: 3 Frame Status: ['0', '3', '9'] Page Faults: 5
Current: 5 Frame Status: ['0', '3', '5'] Page Faults: 6
Current: 6 Frame Status: ['6', '3', '5'] Page Faults: 7
Current: 3 Frame Status: ['6', '3', '5'] Page Hit
Current: 3 Frame Status: ['6', '3', '5'] Page Hit
Current: 9 Frame Status: ['6', '3', '9'] Page Faults: 8
Current: 2 Frame Status: ['2', '3', '9'] Page Faults: 9
Current: 1 Frame Status: ['2', '1', '9'] Page Faults: 10
Current: 0 Frame Status: ['2', '1', '0'] Page Faults: 11
Current: 3 Frame Status: ['3', '1', '0'] Page Faults: 12
Current: 2 Frame Status: ['3', '2', '0'] Page Faults: 13
Current: 3 Frame Status: ['3', '2', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '0'] Page Hit
Current: 1 Frame Status: ['3', '2', '1'] Page Faults: 14
Current: 9 Frame Status: ['9', '2', '1'] Page Faults: 15
Current: 0 Frame Status: ['9', '0', '1'] Page Faults: 16
Current: 9 Frame Status: ['9', '0', '1'] Page Hit
Current: 4 Frame Status: ['9', '0', '4'] Page Faults: 17
Current: 7 Frame Status: ['9', '7', '4'] Page Faults: 18
Current: 0 Frame Status: ['0', '7', '4'] Page Faults: 19
Current: 1 Frame Status: ['0', '7', '1'] Page Faults: 20
Current: 2 Frame Status: ['0', '2', '1'] Page Faults: 21
Current: 8 Frame Status: ['8', '2', '1'] Page Faults: 22
Current: 0 Frame Status: ['8', '2', '0'] Page Faults: 23
Current: 3 Frame Status: ['8', '3', '0'] Page Faults: 24
Current: 0 Frame Status: ['8', '3', '0'] Page Hit
Current: 2 Frame Status: ['2', '3', '0'] Page Faults: 25
Current: 3 Frame Status: ['2', '3', '0'] Page Hit

Fault Ratio: 0.8125

Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['7', '2', '9'] Page Faults: 3
Current: 0 Frame Status: ['7', '2', '9', '0'] Page Faults: 4
Current: 3 Frame Status: ['3', '2', '9', '0'] Page Faults: 5
Current: 5 Frame Status: ['3', '5', '9', '0'] Page Faults: 6
Current: 6 Frame Status: ['3', '5', '6', '0'] Page Faults: 7
Current: 3 Frame Status: ['3', '5', '6', '0'] Page Hit
Current: 3 Frame Status: ['3', '5', '6', '0'] Page Hit
Current: 9 Frame Status: ['3', '5', '6', '9'] Page Faults: 8
Current: 2 Frame Status: ['3', '2', '6', '9'] Page Faults: 9
Current: 1 Frame Status: ['3', '2', '1', '9'] Page Faults: 10
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Faults: 11
Current: 3 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 3 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 1 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 9 Frame Status: ['3', '2', '1', '9'] Page Faults: 12
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Faults: 13
Current: 9 Frame Status: ['3', '2', '1', '9'] Page Faults: 14
Current: 4 Frame Status: ['3', '2', '1', '4'] Page Faults: 15
Current: 7 Frame Status: ['3', '2', '1', '7'] Page Faults: 16
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Faults: 17
Current: 1 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 8 Frame Status: ['3', '2', '1', '8'] Page Faults: 18
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Faults: 19
Current: 3 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 3 Frame Status: ['3', '2', '1', '0'] Page Hit

Fault Ratio: 0.59375

LFU dla różnej ilości ramek:

```
Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['9', '2'] Page Faults: 3
Current: 0 Frame Status: ['9', '0'] Page Faults: 4
Current: 3 Frame Status: ['3', '0'] Page Faults: 5
Current: 5 Frame Status: ['3', '5'] Page Faults: 6
Current: 6 Frame Status: ['6', '5'] Page Faults: 7
Current: 3 Frame Status: ['6', '3'] Page Faults: 8
Current: 3 Frame Status: ['6', '3'] Page Hit
Current: 9 Frame Status: ['9', '3'] Page Faults: 9
Current: 2 Frame Status: ['2', '3'] Page Faults: 10
Current: 1 Frame Status: ['1', '3'] Page Faults: 11
Current: 0 Frame Status: ['0', '3'] Page Faults: 12
Current: 3 Frame Status: ['0', '3'] Page Hit
Current: 2 Frame Status: ['2', '3'] Page Faults: 13
Current: 3 Frame Status: ['2', '3'] Page Hit
Current: 2 Frame Status: ['2', '3'] Page Hit
Current: 1 Frame Status: ['1', '3'] Page Faults: 14
Current: 9 Frame Status: ['9', '3'] Page Faults: 15
Current: 0 Frame Status: ['0', '3'] Page Faults: 16
Current: 9 Frame Status: ['9', '3'] Page Faults: 17
Current: 4 Frame Status: ['4', '3'] Page Faults: 18
Current: 7 Frame Status: ['7', '3'] Page Faults: 19
Current: 0 Frame Status: ['0', '3'] Page Faults: 20
Current: 1 Frame Status: ['1', '3'] Page Faults: 21
Current: 2 Frame Status: ['2', '3'] Page Faults: 22
Current: 8 Frame Status: ['8', '3'] Page Faults: 23
Current: 0 Frame Status: ['0', '3'] Page Faults: 24
Current: 3 Frame Status: ['0', '3'] Page Hit
Current: 0 Frame Status: ['0', '3'] Page Hit
Current: 2 Frame Status: ['2', '3'] Page Faults: 25
Current: 3 Frame Status: ['2', '3'] Page Hit

Fault Ratio: 0.78125
```

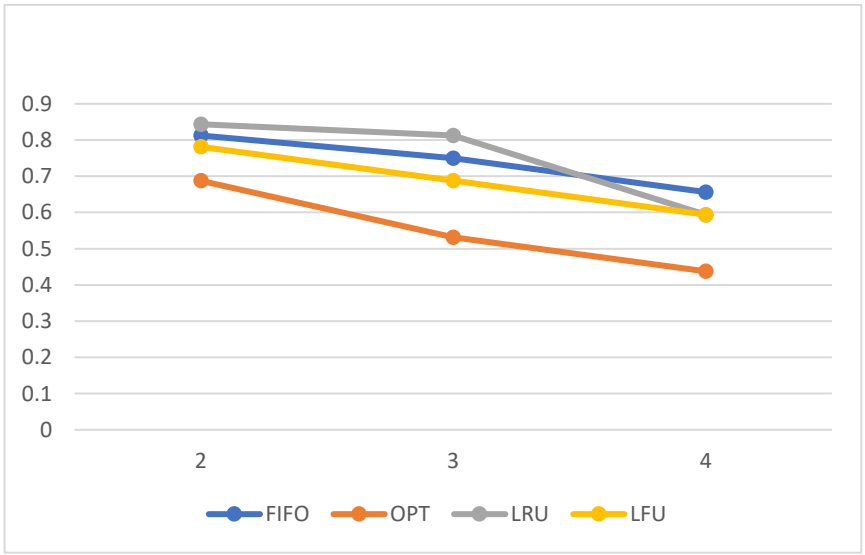
```
Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['7', '2', '9'] Page Faults: 3
Current: 0 Frame Status: ['0', '2', '9'] Page Faults: 4
Current: 3 Frame Status: ['0', '3', '9'] Page Faults: 5
Current: 5 Frame Status: ['0', '3', '5'] Page Faults: 6
Current: 6 Frame Status: ['6', '3', '5'] Page Faults: 7
Current: 3 Frame Status: ['6', '3', '5'] Page Hit
Current: 3 Frame Status: ['6', '3', '5'] Page Hit
Current: 9 Frame Status: ['6', '3', '9'] Page Faults: 8
Current: 2 Frame Status: ['2', '3', '9'] Page Faults: 9
Current: 1 Frame Status: ['2', '3', '1'] Page Faults: 10
Current: 0 Frame Status: ['0', '3', '1'] Page Faults: 11
Current: 3 Frame Status: ['0', '3', '1'] Page Hit
Current: 2 Frame Status: ['0', '3', '2'] Page Faults: 12
Current: 3 Frame Status: ['0', '3', '2'] Page Hit
Current: 2 Frame Status: ['0', '3', '2'] Page Hit
Current: 1 Frame Status: ['1', '3', '2'] Page Faults: 13
Current: 9 Frame Status: ['9', '3', '2'] Page Faults: 14
Current: 0 Frame Status: ['0', '3', '2'] Page Faults: 15
Current: 9 Frame Status: ['9', '3', '2'] Page Faults: 16
Current: 4 Frame Status: ['4', '3', '2'] Page Faults: 17
Current: 7 Frame Status: ['7', '3', '2'] Page Faults: 18
Current: 0 Frame Status: ['0', '3', '2'] Page Faults: 19
Current: 1 Frame Status: ['1', '3', '2'] Page Faults: 20
Current: 2 Frame Status: ['1', '3', '2'] Page Hit
Current: 8 Frame Status: ['8', '3', '2'] Page Faults: 21
Current: 0 Frame Status: ['0', '3', '2'] Page Faults: 22
Current: 3 Frame Status: ['0', '3', '2'] Page Hit
Current: 0 Frame Status: ['0', '3', '2'] Page Hit
Current: 2 Frame Status: ['0', '3', '2'] Page Hit
Current: 3 Frame Status: ['0', '3', '2'] Page Hit

Fault Ratio: 0.6875
```

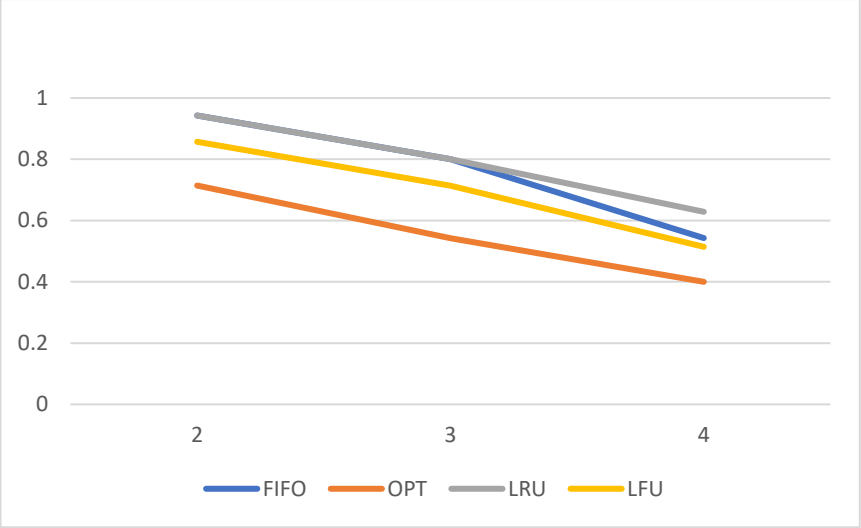
```
Current: 7 Frame Status: ['7'] Page Faults: 1
Current: 2 Frame Status: ['7', '2'] Page Faults: 2
Current: 9 Frame Status: ['7', '2', '9'] Page Faults: 3
Current: 0 Frame Status: ['7', '2', '9', '0'] Page Faults: 4
Current: 3 Frame Status: ['3', '2', '9', '0'] Page Faults: 5
Current: 5 Frame Status: ['3', '5', '9', '0'] Page Faults: 6
Current: 6 Frame Status: ['3', '5', '6', '0'] Page Faults: 7
Current: 3 Frame Status: ['3', '5', '6', '0'] Page Hit
Current: 3 Frame Status: ['3', '5', '6', '0'] Page Hit
Current: 9 Frame Status: ['3', '5', '6', '9'] Page Faults: 8
Current: 2 Frame Status: ['3', '2', '6', '9'] Page Faults: 9
Current: 1 Frame Status: ['3', '2', '1', '9'] Page Faults: 10
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Faults: 11
Current: 3 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 3 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 1 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 9 Frame Status: ['3', '2', '1', '9'] Page Faults: 12
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Faults: 13
Current: 9 Frame Status: ['3', '2', '1', '9'] Page Faults: 14
Current: 4 Frame Status: ['3', '2', '1', '4'] Page Faults: 15
Current: 7 Frame Status: ['3', '2', '1', '7'] Page Faults: 16
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Faults: 17
Current: 1 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 8 Frame Status: ['3', '2', '1', '8'] Page Faults: 18
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Faults: 19
Current: 3 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 0 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 2 Frame Status: ['3', '2', '1', '0'] Page Hit
Current: 3 Frame Status: ['3', '2', '1', '0'] Page Hit

Fault Ratio: 0.59375
```

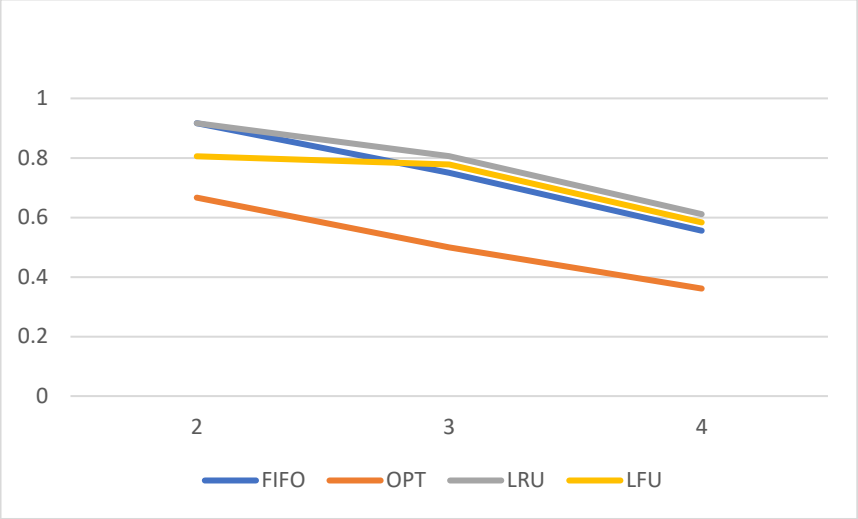
Porównanie Algorytmów

Porównanie algorytmów obejmuje stworzenie losowej próbki danych reprezentujących ciąg odniesień stron do zastępowania oraz wykonanie na ich podstawie algorymów. Na wykresach przedstawiono wyniki dla losowych danych. Oś X przedstawia liczbę ramek a oś Y „współczynnik błędów strony”.

$$Fault\ ratio = \frac{Number\ of\ faults}{Number\ of\ faults + Number\ of\ hits}$$







Na podstawie wykresów można stwierdzić, że algorytm OPT, jako optymalny osiągał najlepsze wyniki. Z pośród pozostałych algorytmów LFU dla dwóch próbek danych okazał się być najlepszy osiągając najmniejszą liczbę błędów stron. FIFO wykazuje najmniejsze różnice niezależnie od próbek i ilości ramek.

Wnioski Dla Algorytmów Zastępowania Stron

Algorytm zastępowania OPT (Optimal) stron rzeczywiście jest optymalny jednakże dla zwiększającej się ilości ramek dla testowanych danych niektóre algorymy (LRU oraz LFU) owocują również optymalnymi wynikami. Chociaż najprostszy w zrozumieniu i implementacji, FIFO zdaje się być tym najgorszym algorymem ponieważ osiąga najgorsze Hit Ratio. Implementacja FIFO może prowadzić do zastępowania stron, które były używane niedawno co można zauważyć porównując FIFO do OPT przy ilości ramek wynoszącej 2. Algorytm OPT wykazuje najlepsze Hit Ratio, ponieważ zakłada pełną wiedzę o przyszłym użyciu stron, co pozwala mu na dokładne przewidzenie, która strona będzie najmniej używana w przyszłości, jednakże z tego względu, w rzeczywistości zazwyczaj jego implementacja jest niemożliwa. Wybór najlepszego algorytmu zastępowania stron zależy od charakterystyki konkretnego systemu oraz rodzaju obciążenia, jakie na nim występuje. Brak jednoznacznej odpowiedzi na to, który algorytm jest najlepszy dla każdego przypadku sprawia, że ważne jest dostosowanie wyboru do konkretnych warunków działania systemu. Również, implementacja każdego z tych algorytmów wymaga uwzględnienia ograniczeń zasobowych i innych czynników specyficznych dla danego środowiska systemowego.