Tomasz Wąsik

PROJEKT Systemy Operacyjne

Algorytmy przydziału czasu procesora oraz algorytmy zastępowania stron

Wstęp teoretyczny:

Projekt napisałem w Pythonie w wersji 3.10.1 i w takiej też zalecam testować.
Link do mojego repozytorium na GitHubie: https://github.com/Wasik-Tomasz/PROJEKT_SO

W moim projekcie porównałem algorytmy przydziału czasu procesora FCFS i SJF.
FCFS (First-Come, First-Served) czyli pierwszy przyszedł i pierwszy zostanie obsłużony oraz SJF (Shortest Job First) najkrótsze zadanie najpierw.
Algorytm FCFS jest najbardziej podstawowym algorytmem i najłatwiejszym do zrozumienia. Wykonuje on wszystkie zadania po kolei, nie biorąc pod uwagę czasu jaki zajmie wykonanie danego zadania. Inaczej za to działa algorytm SJF, ponieważ on bierze pod uwagę czas wykonania każdego zadania i wybiera najkrótszy możliwy z aktualnie dostępnych mu zadań.

W tym projekcie również porównałem algorytmy zastępowania stron FIFO z LRU.
FIFO czyli First In First Out w dosłownym tłumaczeniu pierwsze weszło, pierwsze wyjdzie. Ten algorytm działa na tej samej zasadzie co algorytm FCFS, tylko tym razem odnosi się on do zastępowania stron. Natomiast algorytm LRU (Least Recently Used) polega na zastępowaniu strony, która była najdłużej nie używana.

Opis procedury testowania:

Projekt podzieliłem na trzy główne pliki wykonawcze napisane w Pythonie oraz kilka plików tekstowych. W pliku obsługa_plikow.py posłużyłem się funkcjami w celu zoptymalizowania kodu i poprawienia jego przejrzystości. W nim również stworzyłem „losowanie” danych o konkretnym zadanym ziarnie. Dla większej spójności oba porównania algorytmów są generowane o tym samym ziarnie. W pliku algorytmy_FIFO_LRU.py zastosowałem potrójne pętle for w celu szybkiego obliczenia średnich brakujących stron dla zadanych ramek. Natomiast w pliku algorytmy_FCFS_SJF.py działałem na podwójnych pętlach i zastosowałem funkcję zip w celu posortowania skorelowane danych.

Opracowanie wyników eksperymentu:

Po przeanalizowaniu danych z pliku wyniki.txt stwierdzam, że algorytm SJF jest znacznie wydajniejszy od algorytmu FCFS ze względu na średni czas czekania (331.105 w porównaniu do 494.876). Ciekawą kwestią są średnie przetwarzania, ze względu na to, że są one takie same. Jest to wynikiem tego samego ziarna losowości.

Na postawie wyników z pliku wyniki_stron.txt można zauważyć nieznaczne różnice w wynikach pomiędzy algorytmem FIFO, a LRU. Co ciekawe można zauważyć, że dla ramki o zakresie 3 średnia

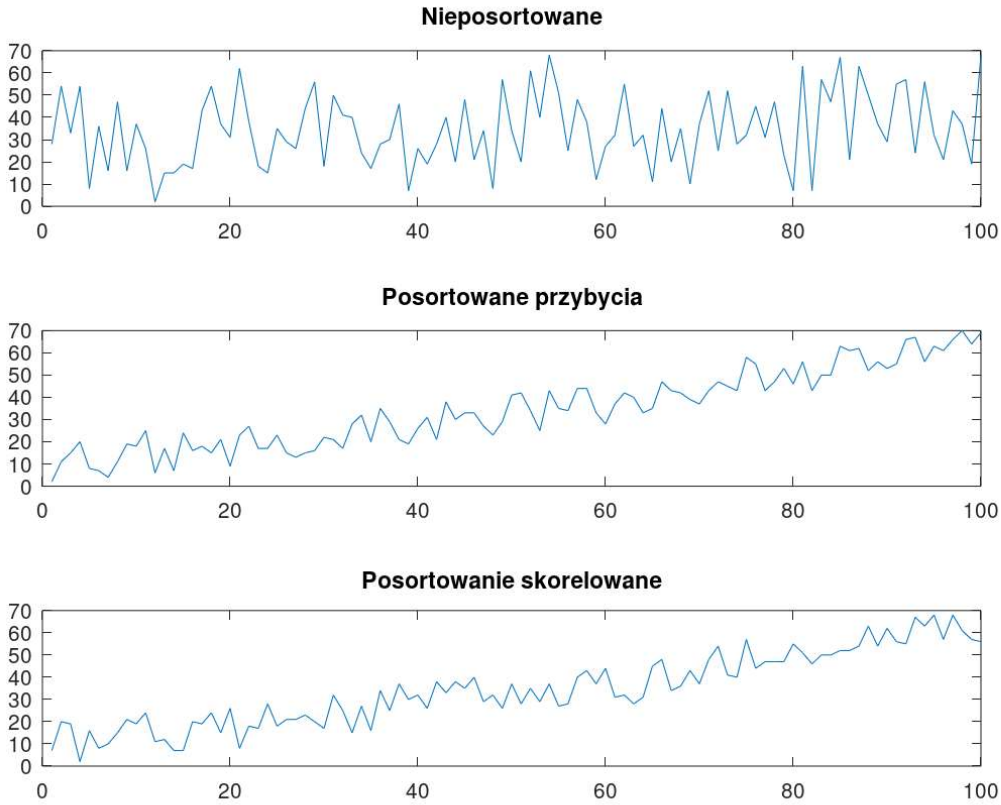
liczba brakujących stron jest mniejsza w algorytmie LRU, jednak dla ramek 5 i 7 zmienia się to i algorytm FIFO ma mniejszą średnią liczbę brakujących stron.

Wnioski:

Wnioski są dość proste, a mianowicie zostało potwierdzone, że pracując, szczególnie na większej ilości danych warto stosować bardziej rozbudowane algorytmy.
Algorytmy FIFO I FCFS można stosować w małych bazach danych, gdzie czas ogólny wykonania zadań nie jest na tyle istotny, aby pokusić się o bardziej rozbudowane algorytmy.

Jako ciekawostkę przedstawiam poniżej wykresy wraz z kodem odnośnie różnicy pomiędzy skorelowanymi moimi liczbami, a brakiem powiązania w trakcie sortowania. Zostało to zrobione z ciekawości ze względu na to, że początkowo nie powiązałem takowych danych i wyniki mogły się różnić, jednak jak widać na wykresach różnica jest marginalna.

Ta sytuacja wystąpiła przy porównywaniu algorytmów FCFS i SJF.



```
przybycia = [27, 44, 19, 35, 1, 30, 14, 39, 1, 24, 6, 1, 3, 13, 1, 7, 31, 45, 22, 28, 46, 18, 8, 6, 20, 23, 22,
39, 50, 7, 42, 37, 25, 6, 11, 8, 17, 42, 5, 18, 6, 25, 21, 10, 35, 9, 29, 7, 50, 15, 1, 50, 38, 49, 41, 16, 29,
20, 5, 25, 22, 40, 14, 27, 5, 26, 6, 23, 2, 32, 44, 13, 44, 23, 13, 29, 27, 40, 10, 1, 48, 6, 49, 40, 48, 4, 45,
43, 26, 22, 48, 39, 5, 48, 18, 9, 26, 17, 5, 49];
wykonania = [1, 10, 14, 19, 7, 6, 2, 8, 15, 13, 20, 1, 12, 2, 18, 10, 12, 9, 15, 3, 16, 20, 10, 9, 15, 6, 4, 5,
6, 11, 8, 4, 15, 18, 6, 20, 13, 4, 2, 8, 13, 3, 19, 10, 13, 12, 5, 1, 7, 19, 19, 11, 2, 19, 10, 9, 19, 18, 7, 2,
10, 15, 13, 5, 6, 18, 14, 12, 8, 5, 8, 12, 8, 5, 19, 16, 4, 7, 13, 6, 15, 1, 8, 7, 19, 17, 18, 7, 11, 7, 7, 18,
19, 8, 14, 12, 17, 20, 14, 19];

przybycia_sort = sort(przybycia);

m = [przybycia;wykonania];
m = rot90(m,1);
m = sortrows(m,1);
m = rot90(m,3);
m = [fliplr(m(1,:));fliplr(m(2,:))];

subplot(3,1,1)
plot (przybycia+wykonania)
hold on
title("Nieposortowane")
hold off

subplot(3,1,2)
plot (przybycia_sort+wykonania)
hold on
title("Posortowane przybycia")
hold off

subplot(3,1,3)
plot (m(1,:)+m(2,:))
hold on
title("Posortowanie skorelowane")
hold off

print("wykres.png")
```