Wprowadzenie do systemów komputerowych
Sprawozdanie z projektu

Zadanie 1 – symulacja planowania czasu procesora

Wybrane algorytmy:

1. FCFS
2. SJF (nie wywłaszczający)
3. SJF (wywłaszczający)

Kryterium Oceniania algorytmów:

Jako kryterium oceniania i porównania algorytmów, został przyjęty średni czas oczekiwania procesów (average response time). Jest to kryterium które pozwala ocenić ile czasu dla średniego procesu jest potrzebne na jego wykonanie. Średni czas oczekiwania jest liczony z czasu oczekiwania każdego procesu podzielonego przez liczbę procesów.

Im niższy średni czas oczekiwania tym lepiej.

Przedstawienie danych:

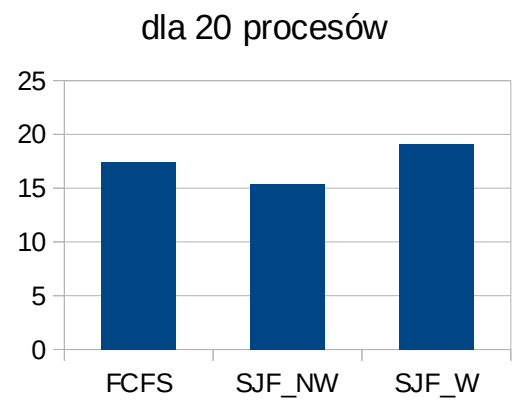
Dane wejściowe składały się z list z różną ilością procesów. Procesy te miały różną złożoność (czas trwania) oraz różny czas przybycia w trakcie symulacji. Dane wejściowe zostały podzielone na różnye typy, gdzie każdy typ charakteryzuje się jakąś konkretną cechą, którą dany zestaw danych odróżnia się od innych. Dane zostały wygenrowane za pomocą skrypu w pythonie dołaczonego do zadania, a ich suche formy znajdują się w katalogu *zad1/datafiles/*

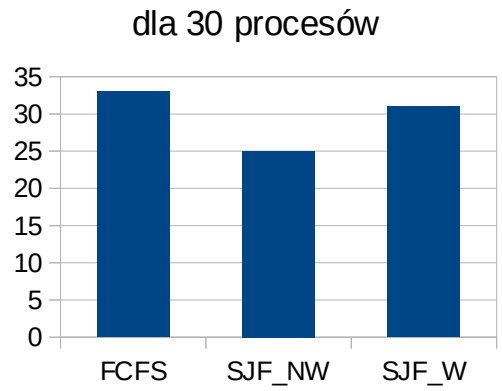
Podział danych na kategorie:

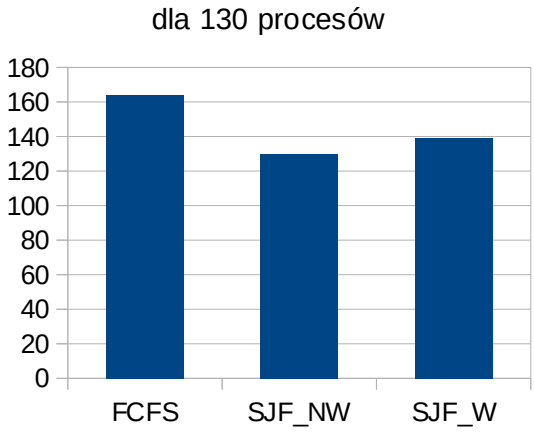
- Po liczbie procesów (mało, dużo)
- Po czasie wykonania procesów (długie, krótkie)
- Po momentach przybycia procesów (małe odstępy, duże odstępy)
- Dane bez kategorii

Dane po liczbie procesów:

Dane zostały przygotowane w taki sposób, że wszystkie procesy trwają pomiędzy 5 a 10, oraz przychodzą w odstępach czasu pomiędzy 7, a 13. Sprawia to, że procesy mało na siebie nachodzą co pozwaa zbadać różnicę w obsłudze dużej i małej ilości procesów. (suche pliki z danymi są nazwane w sposób lp_*, gdzie * to ilość procesów)

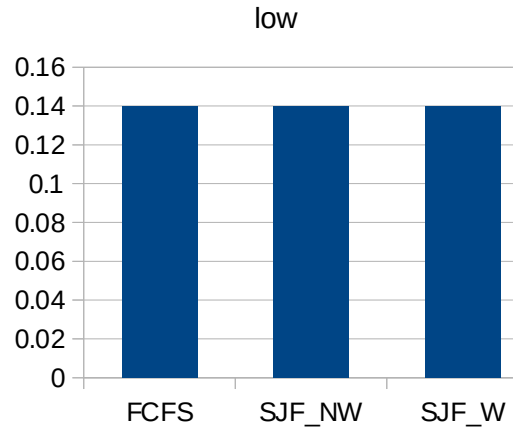


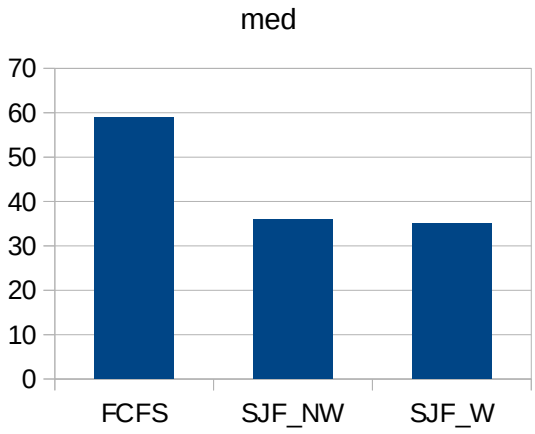


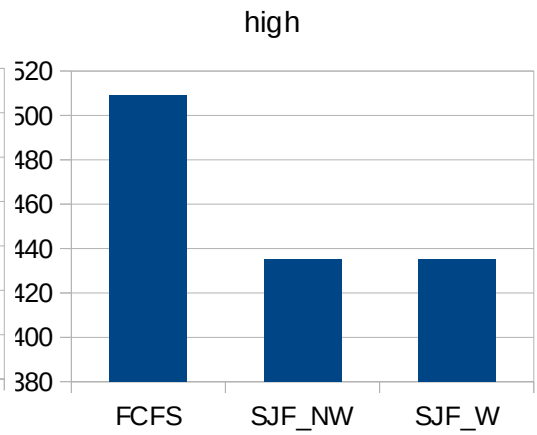


Dane po czasie trwania procesów:

Zostało wygenerowane 100 procesów, które przychodzą w czasie losowym pomiędzy 7 a 13. Dane są podzielone na 3 różne podkategorie gdzie zawierają losowe czasytrwania procesów od 5 do 9 (low), od 15 do 25 (high), oraz od 3 do 20 (med). Taki zestaw pozwala na zaobserwowanie jak na poszczególne algorytmy wpływa czas trwania procesów. (dane są zapisane w plikach pl_*)

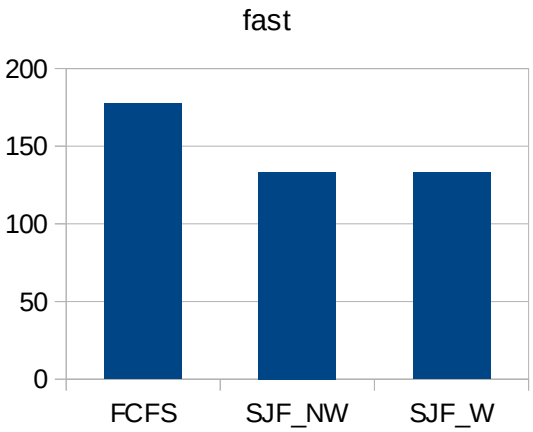


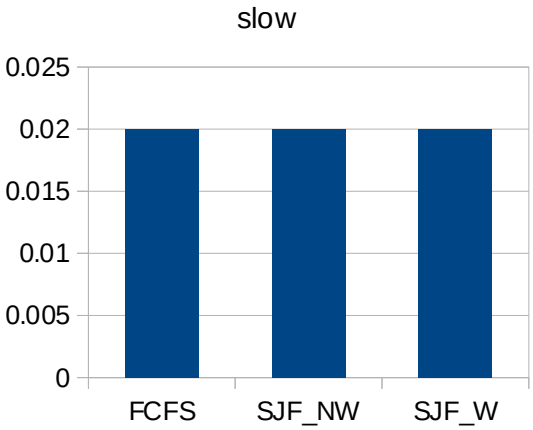




Dane po momencie przybycia procesów:

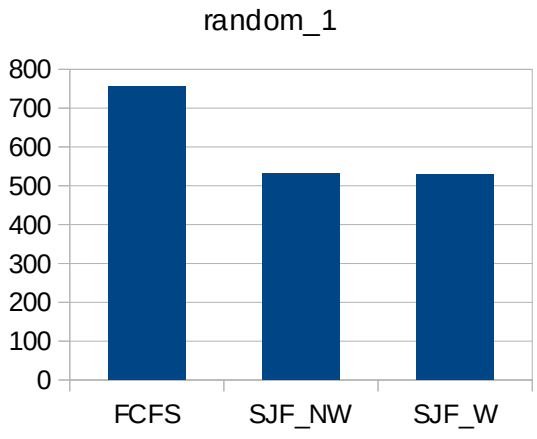
Zostały wygenerowane dwa zestawy dannych po 100 procesów. Każdy proces w każdym z zestawów danych miał czas wykonania pomiędzy 7 a 17. Pierwszy zestaw danych zawiera czasy przybycia od 7 do 10 (plik “fast”), drugi zestaw zawiera czasy przybycia od 15 do 25(plik “slow”) (dane zapisane w plikach at_*)

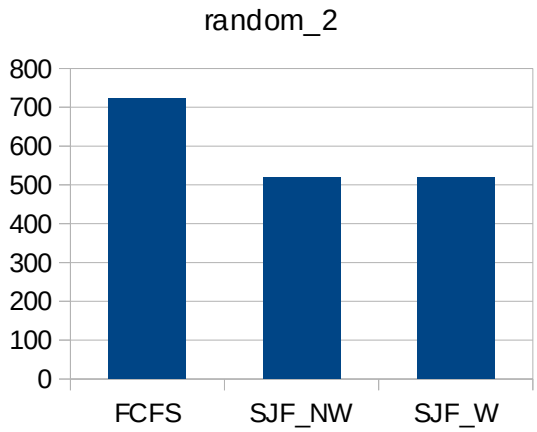




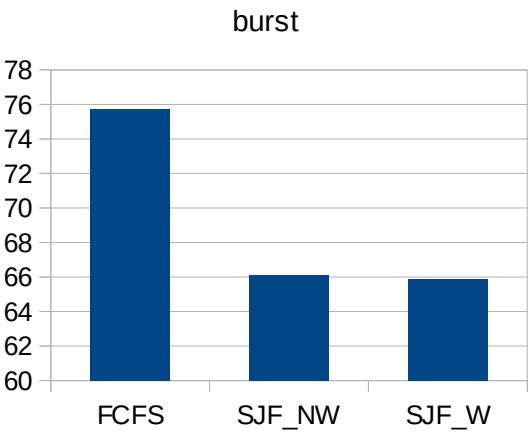
Dane kategorii inne:

zestawy danych “random_1” i “random_2” został przygotowany w taki sposób, że zostało wygenerowanych dużo procesów (200), z parametrami o dużym rozstrzale losowości, a mianowicie czas przybycia od w różnicach od 1 do 30, czas trwania procesu od 10 do 35.





Zestaw danych (burst) zawiera 21 procesów, o czasie trwania od 5 do 15, które znajdują się w 3 bliskich sobie grupach czasowych oddzielonych od siebie interwałami około 30 jednostek czasu.



Wnioski:

Analiza danych wejściowych dla różnych scenariuszy (mała/duża liczba procesów, różne czasy trwania procesów, różne momenty przybycia) pokazuje różnice w efektywności algorytmów:

1. **Liczba procesów:**
    - Przy małej liczbie procesów, różnice w średnim czasie oczekiwania są mniej zauważalne, ale FCFS zwykle wypada najgorzej.
    - Przy dużej liczbie procesów, SJF (wywłaszczający) znacznie przewyższa inne algorytmy, szczególnie FCFS.
2. **Czas trwania procesów:**
    - Dla krótkich procesów, SJF (niewywłaszczający) i SJF (wywłaszczający) radzą sobie znacznie lepiej niż FCFS.
    - Dla długich procesów, FCFS powoduje bardzo wysokie czasy oczekiwania w porównaniu do SJF.
3. **Momenty przybycia procesów:**
    - Przy małych odstępach czasowych, SJF (wywłaszczający) minimalizuje średni czas oczekiwania bardziej efektywnie niż inne algorytmy.
    - Przy dużych odstępach, różnice między SJF (niewywłaszczającym) a wywłaszczającym są mniej wyraźne, ale nadal obecne.

FCFS - (First-Come, First-Served)

Zalety:

- Prosty do implementacji.
- Przewidywalne działanie, łatwe do zrozumienia.

Wady:

- Może prowadzić do problemu konwoju (convoy effect), gdzie długi proces blokuje krótsze procesy, co znacząco zwiększa średni czas oczekiwania.

SJF_NW - (Shortest Job First niewywłaszczający)

Zalety:

- Z reguły minimalizuje średni czas oczekiwania, ponieważ krótsze procesy są obsługiwane szybciej.
- Dobry w środowiskach, gdzie czasy wykonania procesów są znane z góry.

Wady:

- Trudny do wdrożenia w rzeczywistych systemach, ponieważ wymaga znajomości czasu wykonania każdego procesu z góry.
- Może prowadzić do problemu głodzenia (starvation) długich procesów, które mogą być nieustannie odkładane na później.

SJF_W - (Shortest Job First wywłaszczający)

Zalety:

- Dodatkowo zmniejsza średni czas oczekiwania w porównaniu do wersji niewywłaszczającej.
- Elastyczniejszy w obsłudze dynamicznych zmian w kolejce procesów.

Wady:

- Wymaga mechanizmów do wywłaszczania i przełączania kontekstu, co zwiększa złożoność implementacji.
- Może prowadzić do nadmiernego przełączania kontekstu, co negatywnie wpływa na ogólną wydajność systemu.

Podsumowanie

Spośród trzech analizowanych algorytmów, **SJF (wywłaszczający)** wykazuje się najlepszymi wynikami pod względem minimalizacji średniego czasu oczekiwania procesów, szczególnie w środowiskach o dużej liczbie procesów i zróżnicowanych czasach przybycia. **FCFS**, mimo swojej prostoty, okazuje się najmniej efektywny, szczególnie w scenariuszach z dużą liczbą procesów. **SJF (niewywłaszczający)**, choć lepszy od FCFS, nie dorównuje wersji wywłaszczającej pod względem elastyczności i efektywności.

Najlepszy wybór to więc **SJF (wywłaszczający)**, który zapewnia najkrótszy średni czas oczekiwania, szczególnie w dynamicznych i zróżnicowanych środowiskach.

Zadanie 2 - symulacja algorytmów zastępowania stron

Wybrane algorytmy:

- LFU – least frequently used
- LRU – least recently used

Kryterium Oceniania:

Kryterium oceny algorytmu jest częstotliwość występowania błędu stronicowania (page fault) co wpływa negatywnie na szybkość działania systemu. (im mniejsza ilość błędów stronnicowania tym lepiej)

Przedstawienie danych:

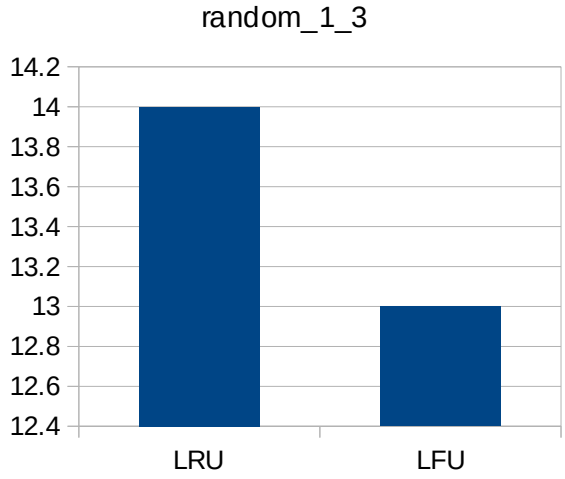
Algorytmy były testowane za pomocą różnych ciągów dostępu do stron. Ciągi te składały się z różnego rodzaju kolejności i wzorów liczb które miały za zadanie zasymulować różne wzorce dostępów do pamięci. Ponadto były także specyfikowane różne wielkości ramek.

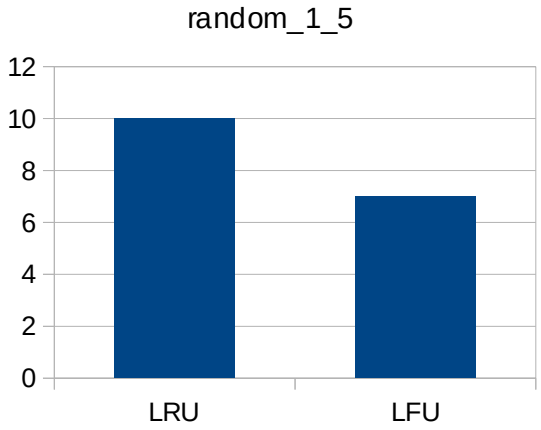
Dane testowe można podzielić na różne kategorie:

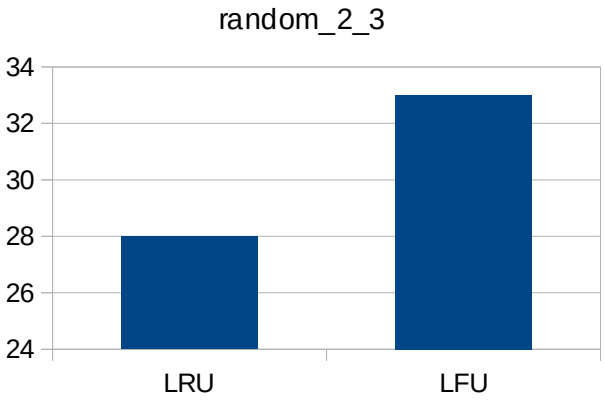
- Testy z losową kolejnością stron
- Test z cykliczną kolejnością stron
- Test z lokalnością odniesień
- Test z dużą ilością stron i małą wielkością ramki

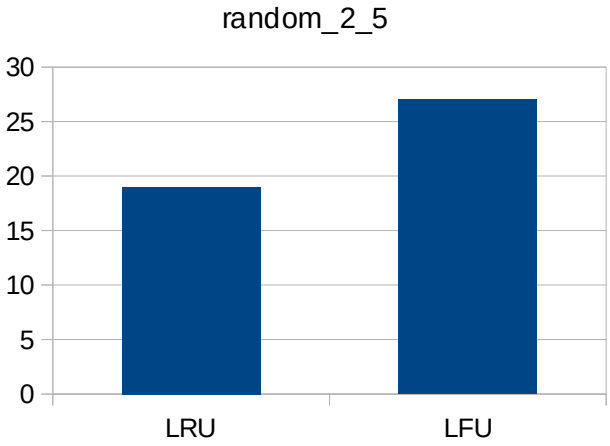
Dane z losową kolejnością stron:

Zostały wygenerowane 2 zestawy danych. W teście pierwszym zostało wygenerowanych 20 stron, z maksymalną ilością różnych stron 8. W teście drugim były zostały wygenerowane 40 stron z maksymalną ilością różnych stron 9. Testy zostały przeprowadzone dla rozmiaru ramki 3 oraz 5. W testach zostało zapewnione, że żadna strona nie wystąpi dwa razy po sobie. (nazwy plików to: rand_x, gdzie x to numer testu) (nazwy wykresów to rand_x_y gdzie x to nr. testu a y wielkość ramki)



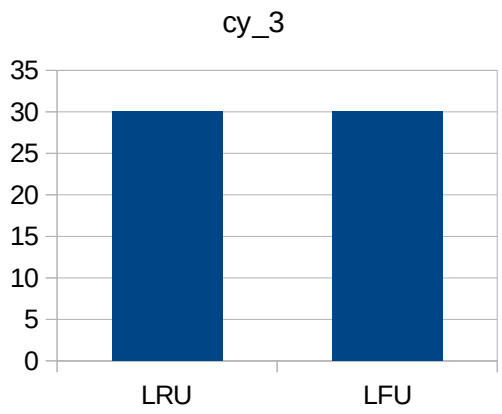


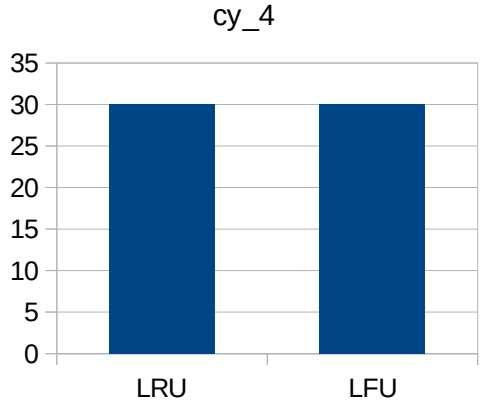


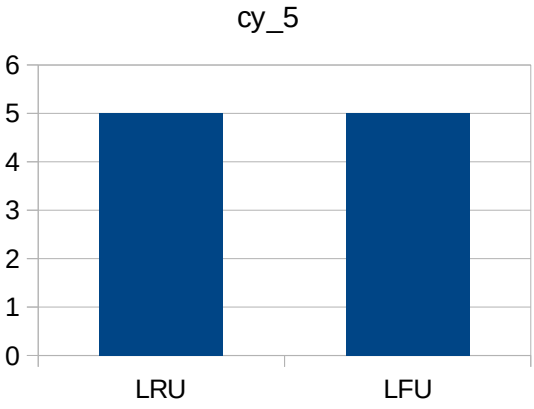


Dane z cykliczną kolejnością stron:

Został wygenerowany test 30 stron, który składa się z sekwencji 5 ramek powtórzonej 6 razy. Test został przeprowadzony dla 3, 4 i 5 stron, żeby pokazać jak algorytmy radzą sobie z cyklicznością oraz jaki wpływ ma ilość ramek na wyniki testów, gdy dane są cykliczne. (nazwy pliku testowego to cy) (nazwy wykresów to cy_x, gdzie x to ilość stron.)

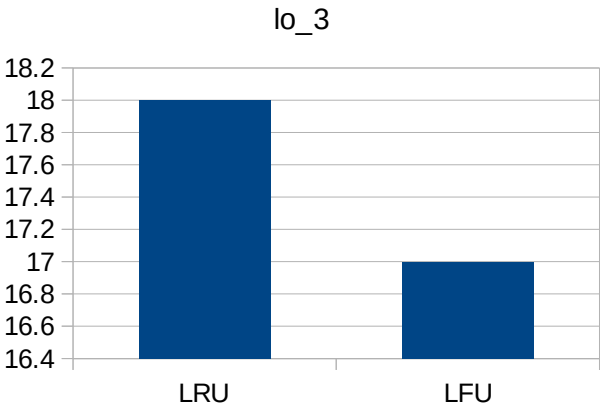


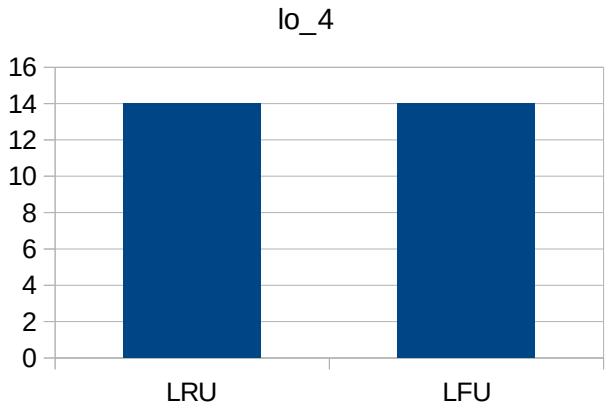




Dane z lokalnością odniesień:

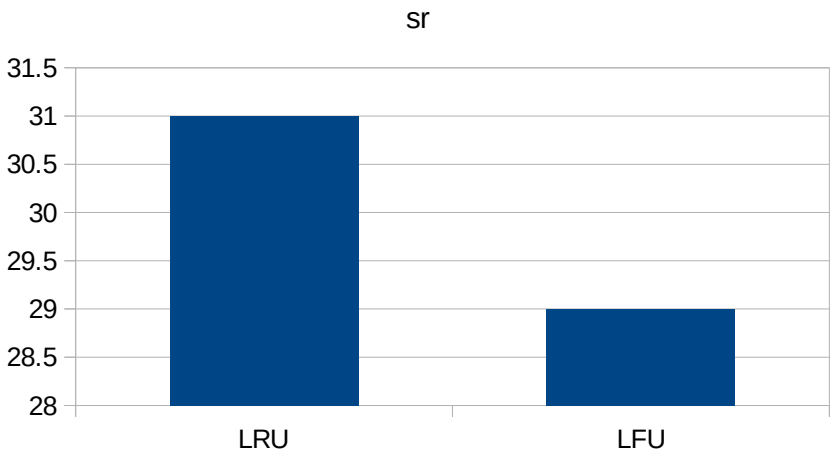
Dane składają się z 30 różnych stron (od 1 do 6), w której niektóre strony pojawiają się częściej niż inne. test został przeprowadzony dla ramki o wielkości 3 i 4. (nazwa pliku lo) (nazwa wykresu lo_x, gdzie x to wielkość ramki)





Dane z dużą ilością stron i małą wielkością ramki:

Dane składają się z 40 stron, gdzie różnorodność stron wynosi od 1 do 9, a wielkość ramki to tylko 3 (mniej niż połowa). W danych zostało zapewnione, że strona nie może występować dwa razy pod rząd. (nazwa danych i wykresu to: sr)



Podsumowanie i Wnioski:

W testach z losową kolejnością stron LFU przeważnie generuje mniej błędów stronicowania w przypadku małej ilości stron niż LRU, jednak przy większej ilości stron LRU wychodzi znacząco na prowadzenie co sugeruje, że LRU lepiej radzi sobie w przypadkach losowego dostępu do pamięci. W testach z cyklicznością stron, algorytmy radzą sobie w dokładnie ten sam sposób, jednak można zauważyć ciekawą zależność od ilości ramki, gdy wielkość ramki równa się cyklowi, ilość błędów znacząco spada. Przy danych z lokalnością odniesień, algorytmy zachowują się dosyć podobnie, jednak przy mniejszej wielkości ramki LFU wysuwa się na nieznaczne prowadzenie przy mniejszej ilości ramek. Przy danych z dużą ilością stron i małą wielkością ramki, algorytm lru wygrywa.

Wybór algorytmu zależy od specyfiki aplikacji i wzorców dostępu do pamięci. W aplikacjach z dużą ilością losowych stron algorytm LRU sprawdzi się najlepiej. Za to w przypadkach gdy strony są bardziej uporządkowane algorytm LFU sprawdzi się lepiej. W przypadku cykliczności danych, większe znaczenie ma wielkość ramki niż dobór algorytmu.

Adnotacja dla sprawdzającego:

Pliki z kodem źródłowym zadania 1 i zadania 2 są zlokalizowane odpowiednio w dwóch folderach zad1/ i zad2/. Pliki z danymi testowymi znajdują się w folderach do każdego zadania w folderze o nazwie datafiles, a pliki z wynikami testów znajdują się w folderach testresults (pod nazwą: res_<nazwa_pliku_z_danymi>). Żeby uruchomić zadanie pierwsze, (do obu zadań trzeba mieć przygotowane środowisko języka rust) należy wpisać komendę:

cargo run <nazwa_pliku_z_danymi>

Żeby uruchomić drugie zadanie trzeba wpisać komendę:

cargo run <nazwa_pliku_z_danymi> <liczba_całkowita_wielkości_ramki>

(każdy program po uruchomieniu, jeżeli został uruchomiony dobrze od razu podaje wyniki dla podanych danych)

W obu folderach z zadaniami zostały także zawarte skypty w języku python do generowania danych.