Wprowadzenie do systemów komputerowych.

Programy symulacyjne.

Maria Filipczak 279449

Cyberbezpieczeństwo grupa piątek 14:15-17:00

Zad. 1 Algorytmy planowania czasu procesora.

W tym projekcie zaimplementowałam dwa algorytmy planowania czasu procesów oraz prosty generator symulacyjnych procesów. Do zadania wykorzystałam algorytm FCFS oraz SJF.

FCFS (First Come First Serve) – w tym algorytmie o kolejności wykonywania procesów decyduje ich czas przybycia, a dokładniej to który z nich nadejdzie jako pierwszy. Nie obsługuje on żadnych priorytetów, a jego prostota potrafi być użyteczna w niektórych sytuacjach.

SJF (Shortest Job First) – w tym algorytmie o kolejności wykonywania procesów decyduje to, który z nich ma najkrótszy czas wykonania. Musimy jednak pamiętać, że aby można było sprawdzić taki czas wykonania jakiś proces musi już nadejść, co za tym idzie ten algorytm jest lekko bardziej skomplikowany niż FCFS jednak również można znaleźć dla niego wiele zastosowań.

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

Na powyższych zdjęciach przedstawiony jest szczegółowy przykład wykonywania czterech procesów o czasie wykonania i przybycia z przedziału od 1 do 100. Pokazują one, że w przypadku algorytmu FCFS posortowane według czasu przyjścia procesy oczekują na swoją kolej gdy inny proces się wykonuje. Kolejka ta wciąż zachowana jest według czasu przybycia. Natomiast w przypadku algorytmu SJF proces po przybyciu zaczyna się wykonywać jeśli ma wolne miejsce, a gdy inne z nich zaczynają się kolejkować sprawdzany jest ich czas wykonywania. Z kolejki wychodzą jako pierwsze najszybsze do wykonania procesy.

* Eksperyment 1

W pierwszym eksperymencie mamy 150, 100, 50 procesów o czasach przybycia i wykonania wylosowanych z zakresu od 1 do 50. Jednostką czasu w naszym przypadku to jedno przejście pętli głównej w programie. Mierzę czas wykonywania procesów, średni czas wykonywania jednego procesu oraz średni czas oczekiwania procesu.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

W przedstawionym eksperymencie widzimy, że czas wykonania procesów jest równy, co za tym idzie w naszym przypadku przez określenie jednostki czasu jako przejście przez pętle, średni czas wykonania jednego procesu również jest taki sam dla obu algorytmów. Ciekawą wartością jednak jest tutaj średni czas oczekiwania, którego różnica między dwoma algorytmami jest dosyć duża w szczególności w ostatnim przypadku gdzie wynosi ona aż około 592 jednostek czasu.

* Eksperyment 2

W drugim eksperymencie mamy 150, 100, 50 procesów o czasie wykonania o czasie przybycia wylosowany z zakresu od 1 do 100 i wykonania wylosowany z zakresu od 1 do 10.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

W przypadku większej ilości procesów o długim czasie przybycia różnica czasu oczekiwania między SFJ i FCFS znacznie maleje nawet dla większej ilości procesów. Wynika to z faktu, że skoro mamy znacznie większe czasy przybycia niż wykonania SFJ jest w stanie sprawdzać mniejszą ilość procesów pod względem czasu wykonania. Mniejsza liczba procesów do porównania oznacza, że SJF ma mniej korzyści z wybierania najkrótszych zadań, ponieważ różnice w długości zadań są mniej istotne.

* Eksperyment 3

W kolejnym eksperymencie mamy 150, 100, 50 procesów o czasie wykonania o czasie przybycia wylosowany z zakresu od 1 do 10 i wykonania wylosowany z zakresu od 1 do 100.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

W przypadku większej ilości procesów o długim czasie wykonania różnica czasu oczekiwania między SFJ i FCFS staje się jeszcze większa. Dodatkowo przy liczbie procesów 150 większa efektywność algorytmu SFJ staje się najbardziej widoczna z wszystkich przeprowadzonych dotychczas eksperymentów biorąc pod uwagę, że różnica między średnim czasem oczekiwania procesu wynosi około 1 056 jednostki czasu.

* Eksperyment 4

W czwartym eksperymencie mamy 150, 100, 50 procesów o czasie wykonania o czasie przybycia z wylosowanych przedziału od 1 do 50 i czasie wykonania 10 równym tyle samo dla każdego procesu.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

W tym eksperymencie średni czas wykonywania jest równy dla wszystkich dla obu algorytmów, ponieważ przy takiej próbie gdy wszystkie procesy mają równy czas wykonywania nie jest wykorzystana możliwość algorytmu Shortest Job First, co za tym idzie w obu przypadkach procesy wykonywane są jak przy FCFS.

**Wnioski**

Po analizie przedstawionych wyników możemy wnioskować, że przez pryzmat mniejszego czasu oczekiwania to algorytm SJF jest lepszy i bardziej opłacalny do wykorzystania, w szczególności w przypadku większej ilości procesów do przetworzenia, gdyż wtedy zauważamy największe różnice. Jednakże FCFS może być używany w systemach, gdzie kolejność wykonywania zadań musi być zgodna z kolejnością przybycia, np. w systemach kolejkowych zadań o równym priorytecie, dodatkowo jest prostszy w implementacji. Oba algorytmy nie są idealne, dodatkowo w przypadku SJF pozostaje pytanie jaką wersję wybieramy w moim projekcie zaimplementowałam wersję bez wywłaszczania, jednak jeśli spojrzymy na SJF z wywłaszczaniem, tym bardziej brakuje w nim „sprawiedliwości”, która występuje w przypadku algorytmu First Come First Served.

Zad. 2 Algorytmy zastępowania stron.

W tym projekcie zaimplementowałam dwa algorytmy zastępowania stron oraz prosty generator symulacyjnych stron, które występują pod postacią listy. Do zadania wykorzystałam algorytm FIFO i LRU.

FIFO (First In First Out)- ten algorytm kieruje się zasadą, która polega na tym, że w przypadku pełnej pamięci usuwana jest z niej najstarsza strona, a na jej miejsce wchodzi najnowsza strona, która została najpóźniej załadowana.

LRU (Least Recently Used)- ten algorytm kieruje się zasadą, która polega na tym, że w przypadku zapełnionej pamięci usuwana jest z niej najdłużej nieużywana strona, a na jej miejsce wchodzi nowa strona, która została najpóźniej załadowana, a co za tym idzie została dopiero co użyta.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, typografia

Opis wygenerowany automatycznie

Powyższe zdjęcia przedstawiają szczegółowy przykład działanie obu algorytmów, główną różnice widzimy przy drugim wypadku, w którym nie występuje błąd, gdzie w drugim algorytmie 9 tym samym zostaje ,,odświeżona” i znowu jest najwcześniej użytą stroną, a za to w przypadku FIFO 9 nie przeszła na początek nie zmieniając kolejki zamiany stron.

* Eksperyment 1

W pierwszym eksperymencie wygenerowałam 100 liczb z przedziału liczb 0-9 imitujących strony. Pojemność pamięci w tym eksperymencie wybrałam na 4.

Algorithm FIFO Hit page number: 38 Page fault number: 62

Algorithm LRU Hit page number: 38 Page fault number: 62

W tym przypadku ilość sukcesów i błędów jest identyczna rośnie ona również dość równomiernie, może to świadczyć o tym, że przy doborze małej pojemności pamięci i takiej losowej kombinacji liczb oba algorytmy dały podobną wydajność dlatego konieczne są dodatkowe eksperymenty.

* Eksperyment 2

W drugim eksperymencie wygenerowałam 150 liczb z przedziału liczb 0-9 imitujących strony. Pojemność pamięci w tym eksperymencie wybrałam na 8.

Algorithm FIFO Hit page number: 117 Page fault number: 33

Algorithm LRU Hit page number: 124 Page fault number: 26

W tym eksperymencie znacznie lepiej widać, że LRU ma mniej błędów, a ilość sukcesów wzrasta szybciej niż w przypadku FIFO. Algorytm FIFO, choć prostszy w implementacji, może prowadzić do większej liczby błędów stron, co może wpływać na wydajność systemu, dlatego w tym przypadku LRU okazał się bardziej efektywny.

* Eksperyment 3

Ciekawym zjawiskiem dotyczącym algorytmów wymiany stron jest Anomalia Belady’yego. Możemy na nią napotkać gdy w przypadku zwiększenia ilości miejsca pamięci uzyskujemy większą ilość błędów niż przed jej zwiększeniem. Niestety mimo kilku prób nie udało mi się wygenerować takiego przykładu. Najbliżej był wypadek wygenerowania 50 stron i zwiększenie pojemności pamięci z 3 do 4 wtedy w algorytmie FIFO ilość błędów i sukcesów pozostała niezmienna. To pokazuje, że nie jest to prosty wypadek do osiągnięcia i nie występuję na ogromną skalę. Jednak chcąc zbadać ten temat zaimplementowałam opcję sprawdzenia takiej anomalii dla konkretnej listy (pages\_anomaly = [1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5]) oraz dla pojemności pamięci 3 i 4.

Obraz zawierający tekst, zrzut ekranu, Czcionka, typografia

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, typografia

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, typografia

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, typografia

Opis wygenerowany automatycznie

Jak widać przedstawiona symulacja tej anomalii działa jedynie na algorytm FIFO, ponieważ liczba błędów stron zwiększyła się o jeden pomimo zwiększenia pamięci. Jest to z pewnością ciekawe zjawisko, jednak jak pokazała próba wywołania go losowo, nie zachodzi ono zawsze dla tego algorytmu, ale warto mieć na uwadze taką sytuację podczas wykorzystywania modelu First In First Out.

**Wnioski**

Po analizie wyników możemy wywnioskować, że oba te algorytmy działają dość podobnie, FIFO jest na pewno zdecydowanie łatwiej zaimplementować, ale LRU nie jest za to w ogóle podatny na anomalie Belad’yego. Przy jednym z eksperymentów widzimy też, że LRU ma małą przewagę względem efektywności i mniejszej ilości błędów względem algorytmu FIFO. Na zrobionych przeze mnie wykresach widać, że nie jest to ogromna różnica, jednak w szczególności na drugim eksperymencie widać, że algorytm LRU ma więcej sukcesów w znacznie szybszym czasie niż FIFO, co może stanowić w pewnych wypadkach znaczącą przewagę.