1. Wstęp

Głównym celem tego projektu jest implementacja czterech wybranych algorytmów i programów symulacyjnych, przeprowadzenie prostych eksperymentów porównujących ich działanie. Z pośród możliwych do wyboru algorytmów utworzono programy prezentujące działanie następujących algorytmów:

- FCFS
- SJF (niewywłaszczeniowy)
- FIFO
- LRU

Poniżej przygotowano krótkie wprowadzenie do działania każdego z nich i przedstawiono teoretyczny efekt ich działania.

1.1. Algorytm FCFS

Z angielskiego First-Come, First-Served scheduling, co w wolnym tłumaczeniu oznacza „pierwsze przychodzi – pierwsze zostaje wykonane”. Jest to jeden z najprostszych, niewywłaszczających algorytmów planowania dostępu do procesora: przydział procesora następuje w kolejności "pierwszy nadszedł - pierwszy obsłużony". Algorytm niewywłaszczający to taki, który obsługuje procesy w z góry ustalonej kolejności i w swoich założeniach nie przewiduje zatrzymania obecnie trwającego, w celu rozpoczęcia kolejnego o wyższym priorytecie. Jeżeli więc w trakcie trwania procesu, w kolejce pojawi się nowy proces o wyższym priorytecie to zacznie on być wykonywany dopiero po skończeniu obecnie trwającego. Algorytm planowania metoda FCFS może powodować efekt konwoju wskutek długotrwałego zajmowania procesora przez niektóre procesy, co niekorzystnie wpływa na średni czas oczekiwania procesów w kolejce do procesora.

1.2. Algorytm SJF

Z angielskiego shortest-job-first, co oznacza że procesor zajmie się wykonywaniem najkrótszego procesu z oczekujących w kolejce. Algorytm ten może być zarówno wywłaszczający jak i niewywłaszczający. W ramach przeprowadzanej symulacji działania algorytmów utworzono program przedstawiający działanie algorytmu niewywłaszczającego, czyli podobnie jak FCFS nie może on przerwać aktualnie trwającego procesu. Zaletą algorytmu SJF jest sortowanie procesów względem długości ich trwania. Takie rozwiązanie zapobiega powstawaniu efektu konwoju.

1.3. Algorytm FIFO

Algorytm wykorzystywany przy stronicowaniu pamięci. Stronicowanie to zabieg zarządzania pamięcią, w którym komputer przechowuje i pobiera dane z pamięci dodatkowej do wykorzystania w pamięci podstawowej. Stronicą nazywa się bloki pamięci dodatkowej o jednakowych rozmiarach. Każdy proces otrzymuje określoną liczbę ramek w pamięci fizycznej. Na miejsca ramek wprowadzane są strony z pamięci logicznej. Algorytm FIFO (z ang. First In – First Out) o jest najprostszym algorytmem wymiany stron, zawsze zastępuje tę stronę, która przebywa najdłużej w pamięci operacyjnej (przyszła jako pierwsza więc będzie musiała zostać wymieniona na nową jako pierwsza).

1.4. Algorytm LRU

Kolejny algorytm stronicowania, który działa na zasadzie zwolnienia ramki w której znajduje się najdłużej nieużywana strona i zastąpienie jej nową, świeżo pobraną z pamięci wirtualnej. Działanie to wymaga pozyskania informacji o czasie użytkowania poszczególnych stron.

2. Struktura programów

2.1. Algorytm planowania czasu procesora FCFS

W strukturze programu zwarto zapytanie, czy symulacja ma zostać przeprowadzona na już utworzonych procesach czy procesy maja zostać utworzone od nowa. W pierwszym przypadku do listy procesów przypisana zostaje zawartość pliku z procesami. W przeciwnym razie program prosi użytkowania o podanie informacji na temat procesów, takich jak: liczba procesów, średnią czasów wykonywania procesów i odchylenie standardowe oraz przedział czasów przyjścia procesów. Początkowo każdy proces ma przydzielony czas oczekiwania oraz czas zakończenia jako 0 (wartość ta będzie się zmieniać w trakcie trwania programu). Z wszystkich wygenerowanych procesów została stworzona lista, która następnie zostaje zapisana do pliku (żeby móc skorzystać z tych samych procesów ponownie).

Cała symulacja odbywa się w pętli, która zostanie przerwana dopiero, gdy wszystkie procesy zostaną ukończone (lista aktualnie oczekujących procesów będzie pusta oraz lista zakończonych procesów będzie miała tyle elementów ile jest procesów). Na samym początku pętli program sprawdza czy jakiś proces właśnie nadszedł. W tym celu przeszukuje listę procesów pod względem ich czasu przyjścia i sprawdza czy jest on równy aktualnemu czasowi. Procesy, które spełnią te warunki zostają dodane do kolejki. W tym momencie zaczyna się właściwe działanie algorytmu, więc zwiększamy czas o jedną jednostkę czasu i sprawdzamy jak zachowują się procesy. Kolejne działania są podejmowane tylko, gdy kolejka nie jest pusta. Wraz z upływem jednej jednostki czasu aktualnie wykonywany proces jest o jedną jednostkę czasu bliżej skończenia, dlatego jego czas wykonywania zostaje zmniejszony o jedną jednostkę. Jeżeli po tym zabiegu czas wykonywania procesu wyniesie 0, oznacza to, że proces właśnie się zakończył. Więc, przypisany zostaje mu czas zakończenia równy aktualnemu czasowi. Musimy mu przypisać jego czas oczekiwania. Jeżeli lista procesów zakończonych jest pusta to aktualny proces jest pierwszym wykonywanym procesem, więc jego czas oczekiwania jest równy 0 (pierwszy proces nie czeka). W przeciwnym wypadku wyliczamy jego czas oczekiwania – czas oczekiwania procesu to czas zakończenia procesu poprzedniego z odliczeniem czasu nadejścia aktualnego procesu. Należy, więc sięgnąć do listy procesów zakończonych, wziąć czas zakończenia ostatniego procesu i odjąć od niego czas przyjścia aktualnego procesu. Jeżeli wynik wyjdzie ujemny to proces przyszedł po zakończeniu poprzedniego procesu, więc jego czas oczekiwania jest równy 0. Tak obliczony czas oczekiwania zostaje przypisany do procesu, a proces zostaje dodany na koniec listy procesów zakończonych. Zostaje on również usunięty z kolejki.

Gdy wszystkie procesy się już zakończą, używając pętli ‘for’ zliczamy wszystkie czasy oczekiwania procesów. Pod koniec zostają wyświetlone procesy wraz z ich czasami zakończenia i oczekiwania oraz średnia czasu oczekiwania.

2.2. Algorytm planowania czasu procesora SJF (niewywłaszczeniowy)

W programie przedstawiającym działanie algorytmu SJF rdzeń kodu w znaczącej części jest taki sam jak w przypadku algorytmu FCFS. Różnica następuje w trakcie dodawania procesów do kolejki. Zanim nowe procesy trafią do kolejki, są one dodawane do tymczasowej listy, która po uzupełnieniu jest sortowana według czasów trwania procesów – od najkrótszego do najdłuższego. Następnie posortowane procesy są kolejno dodawane do kolejki. Zabieg stworzenia tymczasowej listy do sortowania procesów zapobiega sortowaniu procesu aktualnie wykonywanego (wtedy algorytm nie byłby niewywłaszczeniowy). Kolejną różnicą jest to, że po zakończeniu trwania jakiegoś procesu, algorytm ponownie sortuje procesy według ich czasu trwania. Pozostała część kodu nie uległa zmianie.

2.3. Algorytm zastępowania stron FIFO

W strukturze programu zwarto zapytanie, czy symulacja ma zostać przeprowadzona na już utworzonym ciągu odniesienia czy ciąg ma zostać stworzony od nowa. W pierwszym przypadku do listy zawierającej ciąg zostaje przypisana zawartość pliku z ciągiem. W przeciwnym razie program prosi użytkowania o podanie informacji, takich jak długość ciągu odniesienia oraz ilość stron, a następnie na podstawie tych informacji generuje ciąg. Stworzony ciąg zostaje zapisany w pliku, tak aby móc go później odtworzyć. Następnie program prosi użytkownika o podanie ilości ramek po czym przechodzi do wykonania algorytmu. Wszystko odbywa się w pętli, która przerywa swoje działanie, gdy wszystkie elementy ciągu zostaną wykorzystane. Najpierw program sprawdza czy jest jakaś wolna ramka. Jeżeli jest to sprawdza czy pierwsza w kolejce strona znajduje się w którejś z ramek. Jeżeli tak, to strona zostaje usunięta z kolejki i nie zostaną podjęte inne działania. W przeciwnym wypadku do wolnej ramki zostaje dodana strona. Jeżeli już wszystkie ramki są zajęte, program również sprawdza czy oczekująca strona znajduje się w którejś z ramek, jeśli tak to zostaje usunięta z kolejki. W przeciwnym wypadku, opróżniamy ramkę z najwcześniej dodaną stroną (element zerowy listy), a do listy dodajemy oczekującą stronę.

2.4. Algorytm zastępowania stron LRU

W strukturze programu zwarto zapytanie, czy symulacja ma zostać przeprowadzona na już utworzonym ciągu odniesienia czy ciąg ma zostać utworzony od nowa. W pierwszym przypadku do listy zawierającej ciąg zostaje przypisana zawartość pliku z ciągiem oraz zostaje utworzona lista zawierająca numery stron i przypisane do nich numery pozycji, na których zostały ostatnio użyte. W przeciwnym razie program prosi użytkowania o podanie informacji, takich jak długość ciągu odniesienia oraz ilość stron, a następnie na podstawie tych informacji generuje ciąg. Stworzony ciąg zostaje zapisany w pliku, tak aby móc go później odtworzyć. Tutaj również zostaje wygenerowana lista ze stronami. Program prosi o podanie ilości ramek i przystępuje do wykonania algorytmu. Początkowo program przeszukuje listę stron w celu odnalezienia numeru oczekującej strony i zapisania jej pozycji. Do strony przypisany zostaje aktualny numer kroku. Jeżeli jakaś ramka jest wolna, a oczekująca strona znajduje się w którejś ramce to usuwamy ją z kolejki. W przeciwnym wypadku, strona zostaje dodana do wolnej ramki i usunięta z kolejki. Jeżeli wszystkie ramki są już zajęte i oczekująca strona jest w którejś z ramek to usuwamy ją z kolejki. Jeśli strony nie ma w żadnej ramce, stworzona zostaje tymczasowa lista, która jest kopią listy ramek, ale z przypisanymi krokami dla każdej strony. Listę zostaje posortowana według numerów kroków, w których poszczególne strony zostały użyte. Następnie program przeszukuje listę ramek, w celu sprawdzenia i zapisania pozycji, na

której znajduje się najdawniej używana strona. Taką stronę usuwamy i dodajemy do ramki stronę z kolejki.

3. Wyniki

3.1. Algorytmy planowania czasu procesora

Przeprowadzono symulację działania algorytmów FCFS oraz SJF. Za pomocą zaimplementowanego generatora losowych parametrów procesów utworzono bazę o następujących parametrach:

- Ilość procesów – 10
- Średni czas trwania – 15
- Odchylenie standardowe – 8
- Zakres czasów przyjścia – 0-5

Wygenerowana losowo baza procesów przedstawia się następująco:

Tabela 1 Losowo wygenerowana baza procesów

L.p.	Czas przyjścia	Czas trwania
1	4	22
2	3	9
3	5	18
4	0	8
5	3	15
6	1	7
7	0	16
8	5	10
9	2	8
10	3	15

Nadane w powyższej tabeli liczby porządkowe nie stanowią odzwierciedlenia faktycznej kolejności wykonywanych przez program procesów, jednak zostały one w tej kolejności wygenerowane do bazy procesów.

Dla wygenerowanej bazy procesów (patrz Tabela 1) wyniki obydwu algorytmów prezentują się następująco:

Tabela 2. Wyniki symulacji dla bazy zdanych z tabeli 1.

L.p.	Algorytm FCFS		Algorytm SJF	
	Czas zakończenia	Czas oczekiwania	Czas zakończenia	Czas oczekiwania
1	8	0	8	0
2	24	8	15	7
3	31	23	23	13
4	39	29	32	20
5	48	36	42	27
6	63	45	57	39
7	78	60	72	54
8	100	74	88	72
9	118	95	106	83

10	128	113	128	102
Średni czas oczekiwania	48,3		41,7	

Dla tej samej bazy uruchomiono program 3-krotnie w celu sprawdzenia powtarzalności wyników. Pozostały one niezmienne.

W dalszej części przeprowadzono symulacje dla 5 różnych baz procesów o zadanych parametrach, których wyniki prezentują się następująco:

Tabela 3. Wyniki symulacji

		Średni czas oczekiwania	
L.p.	Parametry bazy danych	FCFS	SJF
1	Liczba procesów - 21 Średni czas trwania - 37 Odchylenie stand. - 4 Zakres czasu przyjścia – 2-3	357,14	343.43
2	Liczba procesów - 14 Średni czas trwania - 10 Odchylenie stand. - 1 Zakres czasu przyjścia – 2-3	62,71	60,07
3	Liczba procesów - 20 Średni czas trwania - 15 Odchylenie stand. - 0 Zakres czasu przyjścia – 0-5	139,75	139,75
4	Liczba procesów - 30 Średni czas trwania - 10 Odchylenie stand. - 5 Zakres czasu przyjścia – 0-0	146,33	123,7
5	Liczba procesów - 100 Średni czas trwania - 59 Odchylenie stand. - 23 Zakres czasu przyjścia – 0-20	2870,46	2507,59

3.2. Algorytmy zastępowania stron

Przeprowadzono symulację działania algorytmów FIFO oraz LRU. Za pomocą zaimplementowanego generatora losowego ciągu odniesienia utworzono zbiór o następujących parametrach:

- Długość ciągu odniesienia – 20
- Ilość stron – 7
- Liczba ramek - 3

Wygenerowany został następujący ciąg:

7, 5, 4, 6, 3, 6, 5, 5, 3, 4, 3, 5, 3, 1, 3, 2, 4, 7, 6, 2

Dla algorytmu FIFO wyniki prezentują się następująco:

Tabela 4 Wyniki algorytmu FIFO

Krok	Okno			Kolejka
0	-	-	-	7
1	7	-	-	5
2	7	5	-	4
3	7	5	4	6
4	5	4	6	3
5	4	6	3	6
6	4	6	3	5
7	6	3	5	5
8	6	3	5	3
9	6	3	5	4
10	3	5	4	3
11	3	5	4	5
12	3	5	4	3
13	3	5	4	1
14	5	4	1	3
15	4	1	3	2
16	1	3	2	4
17	3	2	4	7
18	2	4	7	6
19	4	7	6	2
20	7	6	2	-

Ten sam ciąg wykorzystano przy symulacji działania algorytmu LRU

Dla algorytmu LRU wyniki prezentują się następująco:

Tabela 5. Wyniki algorytmu LRU

Krok	Okno			Kolejka
0	-	-	-	7
1	7	-	-	5
2	7	5	-	4
3	7	5	4	6
4	5	4	6	3
5	4	6	3	6
6	4	6	3	5
7	6	3	5	5
8	6	3	5	3
9	6	3	5	4
10	3	5	4	3
11	3	5	4	5
12	3	5	4	3
13	3	5	4	1
14	3	5	1	3
15	3	5	1	2
16	3	1	2	4
17	3	2	4	7
18	2	4	7	6
19	4	7	6	2
20	7	6	2	-

Dla ukazania większej skali działania algorytmów, przeprowadzono również symulację dla ciągu odniesienia o długości 1000 elementów. Wyniki te poddano analizie za pomocą programu Microsoft Excel.

4. Wnioski

4.1. Algorytmy planowania czasu procesora

Analizując wyniki symulacji działanie algorytmów FCFS i SJF dla baz procesów o identycznych parametrach, można zauważyć, że wynikowe średnie czasy oczekiwania w większości przypadków są różne. Przypadkiem w którym obydwa algorytmy uzyskają taki sam wynik, jest warunek stałego czasu trwania każdego procesu. Bazę procesów spełniającą ten warunek wygenerowano poprzez nadanie zerowego odchylenia standardowego. Niezależnie od zastosowanego algorytmu, w takim przypadku nie da się w bardziej efektywny sposób zaplanować czasu pracy procesora (przy użyciu testowanych algorytmów).

Obserwując pozostałe wyniki symulacji, w każdym przypadku algorytm SJF osiąga mniejszy średni czas oczekiwania, można więc uznać, że jest on bardziej efektywny i lepiej optymalizuje czas pracy procesora. Różnica ta wynika z porządkowania oczekujących procesów według ich czasu trwania, co zapobiega powstawaniu efektu konwoju.

Dla ostatniej symulacji, testującej bazę 100 procesów różnica w średnim czasie oczekiwania wynosi ponad 300 jednostek czasu na korzyć algorytmu SJF. Biorąc pod uwagę zapotrzebowanie na ilość wykonywanych procesów, taka różnica w czasie wskazuje jak bardzo istotnym elementem planowanie czasu pracy procesora.

4.2. Algorytmy zastępowania stron

W przypadku symulacji działania algorytmów zastępowania stron nie da się wskazać który z nich jest bardziej efektywny na podstawie pomiaru czasu ich działania czy średniego czasu oczekiwania procesów, jak w przypadku algorytmów planowania czasu procesora. Za główny wyznacznik porównawczy algorytmów FIFO i LRU przyjęto ilość błędów stron.

Błędem strony określany jest stan, w którym algorytm dokonuje wymiany strony w ramce. Z teoretycznego punktu widzenia niemożliwym do przewidzenia jakie kolejne strony pojawią się w kolejce. Niemniej algorytm o większej efektywności powinien zawierać więcej powtarzających się okien pamięci. Poddając analizie wyniki działania algorytmów dla ciągu odniesienia o długości 1000, suma powtórzeń wynosi następująco:

Tabela 6. Suma powtórzeń układu okna pamięci dla próby 1000-elementowego ciągu odniesienia.

Algorytm FIFO	Algorytm LRU
398	390

Biorąc pod uwagę samą sumę powtórzeń można by więc odnieść wrażenie, że algorytm FIFO jest bardziej efektywny, gdyż w efekcie jego działania zaobserwowano więcej duplikatów reprezentacji okna pamięci. Wniosek taki jest błędny i niezgodny z teorią. Nie jest ważna sama suma powtórzeń, a długość ich występowania, czyli ilość kroków w trakcie których okno pamięci pozostaje niezmienne. Przeprowadzona analiza wyników dla 1000-elementowego ciągu odniesień jest nieco zbyt obszerna do skutecznego pokazania tego efektu, dlatego za przykład wybrano wyniki zamieszczone w tabelach 4 i 5.

Na podstawie wyników algorytmu FIFO (Tabela 4), możemy zauważyć, że okna powtarzają się w następujących krokach:

- 5, 6 (2 kroki)
- 7, 8, 9 (3 kroki)
- 10, 11, 12, 13 (4 kroki)

W algorytmie LRU (Tabela 5) powtórzenia występują z kolei w krokach:

- 5, 6 (2 kroki)
- 7, 8, 9 (3 kroki)
- 10, 11, 12, 13 (4 kroki)
- 14, 15 (2 kroki)

Widoczna jest większa ilość powtarzających się okien dla algorytmu LRU, co świadczy o jego większej efektywności.