Spis Treści

1. Wstęp

Zagadnienie szeregowania zadań jest ważną częścią badań operacyjnych. Przestawia ono praktyczne problemy które napotyka się zarówno w informatyce przy zarządzaniu czasem procesora, jak i w rzeczywistych systemach produkcyjnych. Zrozumienie podstawowych zasad i podstaw szeregowania znacząco wzbogaca wiedzę z obszaru optymalizacji oraz pozwala efektywnie rozwiązywać zadania NP-trudne.

Szeregowanie zadań w modelu przepływowym jak i gniazdowym jest tematem poruszanym przez studentów na jednym z laboratoriów na wydziale Elektroniki i Technik Informacyjnych. Dostępne programy wspomagające symulację oraz kontrolę tego typu systemów cechują się dużą złożonością, lub nie udostępniają jasnego interfejsu użytkownika który prezentowałby wyniki przeprowadzonego badania.

Celem niniejszej pracy inżynierskiej było stworzenie narzędzia przystosowanego do efektywnego wspierania procesu analizy wyżej wymienionych systemów. Opracowany program udostępnia prosty interfejs użytkownika, jest w stanie tworzyć modele według zdefiniowanych danych wejściowych oraz prezentuje przebieg i końcowy wynik symulacji w przejrzysty sposób. Za jego pomocą można obserwować konsekwencje zastosowania różnych priorytetów kolejek oraz wyznaczać jasne zależności pomiędzy wieloma sposobami przejścia zadań przez dany system.

2. Systemy przepływowe

Problem szeregowania w systemie przepływowym polega na ustaleniu kolejności obsługiwania zadań na zestawie maszyn zgodnie z zadanymi parametrami. Jest to szczególny przypadek systemu gniazdowego w którym kolejność maszyn jest taka sama dla wszystkich zadań. Do parametrów początkowych należą więc wymagane czasy spędzone na poszczególnych maszynach, jak i cel optymalizacji. Najczęściej celem tym jest minimalizacja czasu trwania całego procesu.

2.1 Definicja formalna

Jest *n* maszyn oraz *m* zadań. Każe zadanie składa się dokładnie z *n* operacji. *I*-ta operacja każdego zadania musi być zrealizowana na *i*-tej maszynie. Żadna maszyna nie może w danej chwili wykonywać więcej niż jedną operację. Dla każdej operacji czas wykonania jest zdefiniowany i znany na początku symulacji. Operacje w ramach każdego zadania muszą być realizowane w ustalonej kolejności. Zadania mogą być realizowane w dowolnej kolejności. Problem polega na znalezieniu optymalnej kolejności wprowadzania zadań do systemu tak, aby osiągnąć zadany cel optymalizacji.

2.2 Złożoność obliczeniowa

Problemy szeregowania zadań w systemach przepływowych są w większości przypadków NP-trudne. Dla systemów o dwóch maszynach, oraz w szczególnych przypadkach o trzech maszynach optymalne rozwiązanie minimalizujące całkowity czas przepływu zadań przez system da się uzyskać stosując algorytm Johnsona. Dla ogólnego przypadku nie istnieje algorytm gwarantujący optymalność rozwiązania.

2.2.1 Algorytm Johnsona

Za pomocą tego algorytmu możliwe jest uzyskanie optymalnego rozwiązania problemu szeregowania zadań w systemie przepływowym z dwoma maszynami przy następujących założeniach:

- wszystkie zadania są dostępne od początku symulacji
- wszystkie wymagane czasy przetwarzania są określone
- pomiędzy maszynami jest nieograniczony bufor

Kroki algorytmu przebiegają następująco:

- należy utworzyć dwie listy zadań *L1* oraz *L2*
- zadania dla których czas przetwarzania na pierwszej maszynie jest krótszy niż czas przetwarzania na maszynie drugiej trafiają na listę *L1*,

pozostałe zadania trafiają na listę *L2.* W przypadku takich samych czasów przetwarzania zadnie trafia na dowolną z list.

- jako pierwsze do systemu powinny wejść zadania z listy *L1* uporządkowane rosnąco według czasu przetwarzania na maszynie pierwszej (reguła SPT), a następnie zadania z listy *L2* uporządkowane malejąco według czasu przetwarzania na maszynie drugiej (reguła LPT). W przypadku takich samych czasów przetwarzania kolejność można dobrać dowolnie.

Algorytm ten zapewnia optymalność rozwiązania, dlatego jest on ważnym elementem nauki o szeregowaniu zadań. Istnieją również adaptacje tego podejścia dla większej liczby maszyn polegające na łączeniu wielu stacji w jedno abstrakcyjne centrum przetwarzania aby zredukować dany problem do zadania o dwóch maszynach.

2.2.2 Przykład zastosowania algorytmu Johnsona

W tym podrozdziale zaprezentuję praktyczne zastosowanie wyżej wymienionego algorytmu do wyznaczenia optymalnej kolejności zadań w systemie przepływowym. Początkowy zestaw zadań wygląda następująco:

Z_i	p_{1i}	p_{2i}
1	2	5
2	5	6
3	4	2
4	9	3
5	5	4
6	1	5
7	5	10

Gdzie Z_i reprezentuje odpowiednie zadanie, p_{1i} to czas przetwarzania na maszynie pierwszej, a p_{2i} to czas przetwarzania na maszynie drugiej. Z powyższego zestawu zadań tworzone są dwie listy według zasad algorytmu. Lista pierwsza:

L1		
Z_i	$\mathbf{p_{1i}}$	p_{2i}
1	**2**	5
2	**5**	6
6	**1**	5
7	**5**	10

Lista druga:

L2		
Zi	p_{1i}	$\mathbf{p_{2i}}$
3	4	**2**
4	9	**3**
5	5	**4**

Następnie zadania z listy *L1* szeregowane są według rosnącej liczby p_{1i} (Z_6,Z_1,Z_2,Z_7), a zadania z listy *L2* szeregowane są według malejącej liczby p_{2i} (Z_5,Z_4,Z_3). Ostatnim krokiem algorytmu jest złączenie obu list co daje wynikową, optymalną kolejność wprowadzania zadań do systemu. Wygląda ona następująco:

$$Z_6, Z_1, Z_2, Z_7, Z_5, Z_4, Z_3$$

Uzyskana kolejność pozwala uzyskać całkowity czas obsługi zadań równy 36. Dla porównania stosując regułę FIFO uzyskuje się czas 44, a przy regule LWR jest on równy 41.

3. Systemy gniazdowe

Będące uogólnieniem modeli przepływowych systemy te nie wymagają aby kolejność maszyn dla wszystkich zadań była taka sama. Co więcej, zadania nie muszą wymagać obsługi na wszystkich maszynach w systemie. Dzięki braku tych ograniczeń problemy te są nadzwyczaj złożone oraz wymagają zaawansowanych technik optymalizacji do znalezienia jak najlepszych rozwiązań. Aktualnie stosuje się programy wykorzystujące sztuczną inteligencję oraz uczenie maszynowe aby poszerzyć wiedzę na temat możliwych rozwiązań tego typu problemów.

3.1 Problem nieskończonego kosztu

Jednym z często napotykanych problemów przy analizie rozwiązań zagadnień szeregowania zadań w systemie gniazdowym jest problem nieskończonego czasu symulacji który występuje w określonych warunkach. Zjawisko to nazywa się zakleszczeniem i jest powodowane oczekiwaniem dwóch maszyn na siebie nawzajem. W przypadku wystąpienia zakleszczenia symulacja nie jest w stanie zmienić stanu maszyn powodujących problem przez co może trwać w nieskończoność.

3.2 Różne warianty systemów gniazdowych

Ponieważ systemy te są jednymi z najbardziej znanych problemów optymalizacji, powstało wiele modyfikacji bazowego modelu aby dostosować go do potrzeb reprezentowania różnorodnych zagadnień. Do urozmaiceń podstawowego problemu szeregowania zadań w systemie gniazdowym należą:

- narzucenie wymagania przerwy po obsłudze każdego zadania
- dążenie do niewystępowania czasu bezczynności maszyny
- definiowanie terminów ukończenia zadań oraz minimalizacja ich opóźnień
- wprowadzanie powiązań między zadaniami, wymóg ukończenia jednego zadania przed rozpoczęciem obsługi innego
- definiowanie wymaganych czasów obsługi z określonym prawdopodobieństwem

4. Istniejące symulatory

5. Założenia systemu

Celem niniejszej pracy było stworzenie symulatora systemów obsługi umożliwiającego tworzenie modeli według danych wejściowych oraz prezentującego mechanizmy rządzące przebiegiem zdarzeń w tych systemach. Aplikacja musiała więc spełniać szereg wymagań aby móc stanowić realne wsparcie procesu nauczania na laboratoriach.

5.1 Dane wejściowe

Aby w pełni zaprojektować określony system i analizować jego działanie, użytkownik musiał mieć możliwość modyfikacji następujących parametrów:

- ilość maszyn w systemie
- rozmiar bufora przed każdą maszyną
- ilość zadań w systemie
- wymagany czas zadań na poszczególnych maszynach
- regułę priorytetu dla każdego bufora
  - FIFO (First In First Out)
  - LIFO (Last In First Out)
  - SPT (Shortest Processing Time)
  - LPT (Longest Processing Time)
  - LWR (Least Work Remaining)
  - priorytet własny, przykładowo wykorzystywany w modelowaniu systemu optymalizowanego algorytmem Johnsona.

Dla systemów gniazdowych musiała dodatkowo istnieć możliwość modyfikacji kolejności wymaganych maszyn w obrębie zadania, lub całkowitego usunięcia wymagania określonej maszyny.

5.2 Przebieg symulacji

Po uruchomieniu symulacji program miał za zadanie prezentować jej przebieg w sposób jasny i zrozumiały, aby jak najlepiej wyjaśnić mechanizmy działania systemu. Aby to osiągnąć prezentowane musiały być następujące rzeczy:

- pozycja poszczególnych zadań w systemie
- postęp wykonania zadań na aktualnie zajmowanych przez nie maszynach
- zajętości wszystkich maszyn
- zajętości buforów
- wykres Gantta dla aktualnego stanu symulacji
- zakleszczenia (w przypadku wystąpienia)

Dodatkowo aby umożliwić użytkownikowi większą kontrolę nad przebiegiem symulacji program udostępnia trzy różne metody generowania kolejnych stanów modelu:

- tryb krokowy – stany modelu prezentowane są w odstępie jednej jednostki czasowej
- tryb zdarzeniowy - symulacja zatrzymuje się po wystąpieniu jakiegokolwiek zdarzenia
- tryb natychmiastowy - program wykonuje symulację natychmiastowo i prezentuje jedynie podsumowanie końcowe

5.2 Podsumowanie symulacji

Po zakończeniu symulacji program prezentuje statystyki podsumowujące jej przebieg. Dzięki nim można ocenić jakość zaprojektowanego modelu oraz wprowadzić ewentualne poprawki. Do prezentowanych rzeczy należą:

-

6. Realizacja systemu

Przed rozpoczęciem prac nad systemem konieczna była decyzja z jakich narzędzi skorzystać przy tworzeniu aplikacji. Program musiał być kompatybilny z oprogramowaniem obecnym na sprzęcie w laboratorium, oraz nie mógł polegać na zewnętrznych zasobach które trzeba by instalować na maszynach oraz regularnie aktualizować. Ważnym aspektem była też wydajność, aby symulacja mogła odbywać się w możliwie krótkim czasie nawet przy dużej ilości danych wejściowych. Aplikacja powinna być również stabilna, aby zapewnić możliwość bezproblemowego korzystania z niej podczas trwania laboratorium.

Oprócz wymagań co do funkcjonowania finalnej aplikacji, musiałem też zwrócić uwagę na stopień skomplikowania tworzenia zaawansowanego interfejsu użytkownika w różnych językach programowania. Za zaletę postrzegana była łatwość wprowadzania zmian oraz utrzymania kodu, zarówno jak jego przejrzystość. Wybrane narzędzie musiało również umożliwiać pełną kontrolę nad tworzonymi elementami interfejsu, aby móc implementować własną logikę dynamicznego pojawiania się odpowiednich przycisków oraz pól tekstowych.

6.1 Język programowania

Podstawowym wyborem którego musiałem dokonać był wybór języka programowania. Większość wiodących języków udostępnia biblioteki służące tworzeniu zaawansowanych interfejsów użytkownika, dlatego przeanalizowałem wiele z popularnych narzędzi i zdecydowałem się na skorzystanie z Javy. Poniżej przedstawię niektóre z możliwości jak również ich wady i zalety.

6.1.1 Python

Język ten został zapoczątkowany w latach dziewięćdziesiątych, jednak popularność zaczął szybko zyskiwać dopiero w roku 2004. Aktualnie jest jednym z najpopularniejszych języków programowania ogólnego zastosowania, a liczba korzystających z niego osób wciąż stale rośnie.

Głównymi zaletami Pythona jest przejrzystość i prostota kodu, jak również jego ekspresywność. Jedna linia napisana w Pythonie jest równoważna nawet do 6 linii w języku C. Dzięki temu aplikacje powstają szybko i efektywnie, a sam kod jest czytelny i zwięzły. Język ten jest również częściowo interpretowany co przekłada się na brak dodatkowych wymagań przy korzystaniu z niego na wielu różnych systemach operacyjnych.

Główną wadą jest brak uznanego i wspieranego narzędzia do tworzenia interfejsów użytkownika. Istnieje wiele bibliotek które umożliwiają tworzenie różnych elementów interaktywnych, jednak większość z nich wywołuje się z poziomu kodu, co przy tworzeniu dużej liczby paneli i pól tekstowych staje się wyjątkowo czasochłonne i wymaga dużej uwagi. Aby uruchomić skrypt Pythona na komputerze, wymagana jest uprzednia instalacja odpowiedniego oprogramowania oraz zalecana jest jego regularna aktualizacja, co mogłoby stwarzać problemy na sprzęcie laboratoryjnym.

6.1.2 C++

Wywodzący się z z języka C, stworzony w latach osiemdziesiątych język C++ jest najlepszym wyborem pod względem wydajności i stabilności stworzonego kodu. Stanowi on swojego rodzaju standard w programowaniu przez co dostępne zasoby są zawsze na najwyższym poziomie. W języku tym stosowana jest jednoznaczna składnia z silną kontrolą typów, co pozwala na bardzo szybkie i automatyczne modyfikowanie dużych bloków logiki programu.

Powodem dla którego nie wybrałem tego języka jest jego mocna zależność od obecnych na komputerze bibliotek i wymóg uprzedniej kompilacji programu. Przez to nie jest on łatwo przenośny pomiędzy różnymi systemami operacyjnymi. Ponieważ C++ daje kontrolę programiście nad zarządzaniem pamięcią, tworzenie aplikacji wymaga wyjątkowej uwagi oraz znajomości wykorzystywanych bibliotek. Podobnie do Pythona, tutaj również nie ma uznanego narzędzia do łatwego tworzenia i konfigurowania interfejsów.

6.1.3 Java

Jest to język stworzony w roku 1991. Bardzo szybko zyskał on na popularności, dzięki czemu aktualnie jest najczęściej używanym językiem programowania na świecie. Wynik ten jest skutkiem łatwości uruchamiania aplikacji na różnych systemach operacyjnych, w tym na urządzeniach mobilnych, jak również jednoznaczna struktura kodu oraz reguły zaprojektowane z myślą o automatycznej kontroli możliwych błędów programistycznych.

Największym czynnikiem decydującym o wyborze tego właśnie języka jest dostępność narzędzi do tworzenia aplikacji okienkowych. Z powodu ogromnej popularności

dostępnych jest wiele gotowych programów do tworzenia elementów interfejsu, jak również większość problemów które można napotkać jest szczegółowo omówiona na forach internetowych. Dzięki temu umożliwienie użytkownikowi prostego wprowadzania parametrów wejściowych oraz analizowania wyników było znacznie uproszczone, przez co więcej czasu mogłem poświecić na dopracowanie logiki aplikacji oraz testy.

Jedną z wad Javy która mogła negatywnie wpłynąć na działanie aplikacji jest mniejsza w porównaniu do C++ wydajność. Przez to symulacja przebiega wolniej, jednak różnica jest niezauważalna nawet przy znacznej liczbie maszyn oraz zadań w systemie.

6.2 Środowisko programistyczne

Po zdecydowaniu się na wykorzystanie Javy jako języka do napisania aplikacji, musiałem wybrać odpowiednie środowisko do utworzenia w nim projektu z kodem programu. Spośród wielu popularnych możliwości przeanalizowałem dwa najpopularniejsze środowiska: Eclipse oraz NetBeans, decydując się na wykorzystanie drugiego.

6.2.1 Eclipse

Jest to platforma napisana w 2004 roku. Jej główną zaletą jest rozbudowana biblioteka dostępnych rozszerzeń które dostosowują środowisko do indywidualnych potrzeb użytkownika. Narzędzie to w znacznym stopniu ułatwia zarządzanie dużymi projektami informatycznymi oraz ma wiele wbudowanych funkcjonalności. Główną wadą tego oprogramowania jest jego złożoność. Początkujący użytkownik ma dostęp do wszystkich zaawansowanych funkcji programu przez co wykonywanie prostych czynności może być utrudnione.

6.2.2 NetBeans

Środowisko to ma swoje początki jako projekt studencki zrealizowany w 1996 roku na Uniwersytecie Karola w Pradze. W 2010 roku projekt przeszedł na własność firmy Oracle, co zapoczątkowało gwałtowny wzrost jego popularności.

Najbardziej wartościową dla mnie cechą tego oprogramowania jest wbudowany w niego edytor interfejsu użytkownika. Umożliwia on proste tworzenie elementów interaktywnych poprzez wybieranie ich ze specjalnego zasobnika. Kod odpowiadający za komunikację przycisków i pól tekstowych z logiką programu generowany jest automatycznie, co redukuje czas potrzebny na pisanie programu oraz zmniejsza liczbę potencjalnych błędów. Narzędzie to jest kompaktowe oraz przyjazne użytkownikowi. Wszystkie jego funkcjonalności są łatwo i intuicyjnie dostępne.

6.3 Kontrola wersji

Podczas rozwijania każdego projektu informatycznego niezbędne jest korzystanie z oprogramowania zapewniającego kopie bezpieczeństwa tworzonego kodu. W razie wprowadzenia funkcjonalności powodujących nieprawidłowe działanie programu należy mieć możliwość szybkiego cofnięcia się do wersji stabilnej, jak również analizy wprowadzonych błędów w celu ich naprawy. Problemy te napotykane są często przy tworzeniu komercyjnych aplikacji, dlatego powstało wiele narzędzi ułatwiających prowadzenie kontroli wersji.

6.3.1 GIT

Jest to system początkowo stworzony jako narzędzie wspomagające rozwój jądra Linux. Jego główną zaletą jest zapewnienie wsparcia dla rozgałęzionych procesów tworzenia oprogramowania, co pozwala na równoczesną pracę wielu osób nad systemem, lub równoległe rozwijanie wielu jego funkcjonalności. Dzięki temu jest on aktualnie najpopularniejszym systemem kontroli wersji na świecie.

Podczas pisania aplikacji, GIT stanowił miejsce bezpiecznego przechowywania kopii zapasowych, jak również udostępniał historię zmian projektu, co pozwalało na szybką analizę błędów polegających na niepoprawnie współpracujących ze sobą częściach programu. W celu zabezpieczenia się przed awarią komputera lub dysku twardego, repozytorium projektu utworzyłem w darmowym serwisie internetowym GitHub. Dzięki temu oprócz lokalnych kopii zapasowych istniała też kopia zewnętrzna projektu, która zapewniała brak możliwości utracenia postępu prac.

Jako narzędzie wspomagające korzystanie z wybranego systemu kontroli wersji wykorzystałem program Git Extensions. Komunikacja z serwerem repozytorium odbywa się za pomocą poleceń tekstowych wywoływanych z konsoli, dlatego użycie jednego z wielu narzędzi udostępniających graficzny interfejs użytkownika znacznie skraca czas wymagany na odpowiednie tworzenie kopii projektu jak również poruszanie się po jego rozgałęzieniach.