Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод виявлення взаємного блокування в інформаційних системах

(тема)

Виконав:

студент II курсу, групи КСМм-22-1

Мантуров Д. О.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Комп’ютерні системи та мережі
(повна назва освітньої програми)

Керівник: проф. Горбачов В.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Мантурову Денису Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Метод виявлення взаємного блокування в інформаційних системах

затверджена наказом по університету від “ 06 ” листопада 2023 р. № 1298Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2024

3. Вхідні дані до роботи 1) алгебра процесів; 2) процесні моделі систем з мережевою структурою; 3) тупики в алгебрі процесів.

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз аналітичних моделей систем з мережевою структурою;
2) аналіз аналітичних засобів для опису тупиків;
3) метод виявлення і блокування тупиків.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд-презентація – 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз інформаційних систем та моделей	07.11.23-13.11.23	
2	Постановка задачі	14.11.23-20.11.23	
3	Аналіз засобів алгебри процесів	21.11.23-23.11.23	
4	Розробка методу пошуку та рішення	24.11.23-06.12.23	
	взаємоблокувань		
5	Розробка методу верифікації моделей систем	07.12.23-23.12.23	
6	Оформлення матеріалів кваліфікаційної роботи	26.12.24-02.01.24	
7	Подання кваліфікаційної роботи керівникові та	03.01.24-06.01.24	
	Її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	09.01.24-12.01.24	
	вання		

Дата видачі завдання 06 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Горбачов В.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 82 с., 15 рис., 2 дод., 18 джерел.

ІНФОРМАЦІЙНА СИСТЕМА, ПРОЦЕС, ПОДІЯ, ТУПІКОВА СИТУАЦІЯ, WAIT-FOR ГРАФ, ГРАФ РОЗПОДІЛУ РЕСУРСІВ, OR-МОДЕЛЬ, UML, АЛГЕБРА ПРОЦЕСІВ.

Метою кваліфікаційної роботи є аналіз особливості тупикових ситуацій в інформаційних системах процесів на рівні опису проблеми. У роботі розглянуто існуючі на сьогоднішній день методи пошуку та вирішення тупиків, а також проведено їх аналіз, виділено переваги та недоліки. Досліджено можливість використання теорії алгебри процесів для завдань пошуку та вирішення глухих кутів.

У роботі запропоновано метод вирішення тупиків та перевірки правильності функціонування системи після усунення тупикової ситуації.

ABSTRACT

Master’s thesis: 82 pages, 15 figures, 2 appendices, 18 sources.

INFORMATION SYSTEM, PROCESS, EVENT, DEADLINE, WAIT-FOR GRAPH, RESOURCE DISTRIBUTION GRAPH, OR-MODEL, UML, ALGEBRA OF PROCESSES.

The purpose of the qualification work is to analyze the peculiarities of deadlock situations in information systems processes at the level of problem description. The work examines the currently existing methods of finding and solving dead ends, as well as analyzes them, highlights their advantages and disadvantages. The possibility of using the theory of algebra of processes for the tasks of finding and solving dead ends has been studied.

The paper proposes a method for solving dead ends and checking the correct functioning of the system after eliminating the dead end situation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система

AWFG – асинхронний граф очікування (англ., Asynchronus Wait For Graph)

CCS – обчислення систем, що спілкуються (англ., Calculus of Communicating Systems)

UML – уніфікована мова моделювання (англ., Unified Modeling Language)

LTS – система помічених переходів (англ., Labelled Transition System)

SOS – структурна тксплуатаційна семантика (англ., Structural Operational Semantics)

ВСТУП

Дослідження працездатності складних систем є одним із найважливіших завдань сучасної науки і техніки. Комплексні системи – це складні технологічні, промислові, енергетичні, комунікаційні комплекси, автоматизовані системи управління та високопродуктивні багатопроцесорні обчислювальні системи.

Моделювання широко використовується для аналізу складних систем (CS), але для моделювання CS необхідно описати її в термінах деякої формальної схеми. Існує багато різних схем для опису компонентів складних систем, таких як диференціальні рівняння, кінцева автоматизація, імовірнісна автоматизація, системи масового обслуговування. Але всі вищенаведені схеми можна використовувати для опису компонентів складної системи у вузькій галузі промисловості, оскільки всі вони мають багато надмірностей. Ця робота зосереджена на дослідженні процесу, математичної схеми високого рівня абстракції та моделей складної системи, яка була описана за допомогою процесів.

Розглядаючи великомасштабні складні системи, ми стикаємося з проблемами, пов’язаними з дуже довгим циклом моделювання і винекня тупиків. Одним із способів прискорення процесу моделювання та усунення тупиків є використання розподіленого моделювання на основі багатопроцесорних комп’ютерних систем. Але перед моделюванням необхідно підготувати аналізовану вихідну систему та виявити можливість появи тупиків. Це означає, що вихідна система повинна бути представлена за допомогою типової математичної схеми, а опис системи повинен відповідати архітектурі розподіленого обладнання. Щоб максимізувати ефективність процесу моделювання і усунення тупиків, ми повинні реорганізувати вихідну систему в набір процесів, і кількість процесів має бути співвіднесена з кількістю процесорів. Іншими словами, якщо комп’ютер має «n» процесорів,

нам потрібно реорганізувати вихідну модель і отримати модель, яка включає лише «n» процесів. Було б можливо здійснити таку реорганізацію та отримати можливість усунення тупиків, якби ми змогли виявити властивий паралелізм у структурі моделі та зробити агрегацію процесів відповідно до вибраних частин моделі. У цьому випадку кожна виділена частина буде незалежною, а взаємозв'язки з іншими частинами будуть мінімальними. В результаті цієї операції ми отримаємо "n" процесів, що відповідають "n" процесорам, в той же час локалізуємо тупикову ситуацію і збільшим ймовірність її усунення, при зменшенні обчислювального ресурсу для пошуку та усунення тупиків.

У цій роботі запропоновано алгоритм для виявлення внутрішнього паралелізму в структурі моделі та алгоритм для реорганізації вихідної моделі з метою отримання нової моделі, яка може бути, з одного боку ефективно змодельована, а з другого боку без тупиків, на базі багатопроцесорної комп'ютерної системи. Запропоновані алгоритми реалізовано в програмі, яка працює з оригінальною моделлю та пропонує нову структуру моделі для розподіленого моделювання.

1 АНАЛІЗ ПРОБЛЕМИ ТА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз літератури

1.1.1 Тупики у інформаційних системах

Сьогодні більшість програмних систем складаються з різних частин і компонентів, які повинні працювати паралельно. Отже, ці системи розроблені для підтримки високого ступеня паралельності. Взаємоблокування – це небажаний стан і загроза, яка може виникнути в паралельних системах через спільне використання ресурсів і даних. Отже, при розробці таких систем однією з важливих властивостей, яку необхідно перевірити, є відсутність взаємоблокувань. Добре відомим і повністю автоматичним методом виявлення помилок, таких як взаємоблокування, є перевірка моделі, у якій усі доступні стани системи генеруються з початкової конфігурації. Цей набір станів називається простором станів. Однак для багатьох складних систем простір станів має тенденцію бути занадто великим, щоб бути дослідженим вичерпано. Таким чином, виникає проблема вибуху простору стану [1].

Протягом останніх років, щоб полегшити проблему вибуху простору стану, було використано декілька методів, таких як символічна перевірка моделі, часткове зменшення порядку [2], зменшення симетрії і перевірка моделі на основі сценаріїв [4]. Фактично, вони зменшують необхідну пам'ять, зменшуючи кількість станів, які потрібно дослідити.

Алгоритми оцінки розподілу (EDA) – це клас еволюційних алгоритмів, які замінюють традиційні генетичні оператори, такі як кросинговер і мутація, імовірнісною моделлю перспективних рішень. Насправді ця модель фіксує залежності/незалежності між змінними проблеми. Методи машинного навчання використовуються для отримання відповідної інформації з населення та створення моделі. Потім вивчена модель відбирається для

створення нових рішень, які будуть включені в популяцію. Основна відмінність між різними EDA полягає в типі ймовірнісної моделі. Байєсова мережа (BN) є популярною моделлю в літературі EDA і використовується в різних EDA, які називаються байєсівськими EDA [3, 4]. Можливості навчання та експресивна здатність цих мереж зробили їх популярними та корисними.

У цьому документі, щоб покращити існуючі підходи з точки зору точності та швидкості конвергенції, пропонується новий підхід із використанням BOA для виявлення тупикових ситуацій у складних системах, формально заданих через перетворення графів. Щоб застосувати еволюційні алгоритми оптимізації, такі як GA або BOA, у перевірці моделі, ми повинні визначити структуру хромосом. Для цього в кожному шляху, починаючи з початкового стану, ми визначаємо хромосому як упорядковану послідовність вихідних переходів станів.

У цій статті BOA застосовано до проблеми перевірки моделі. Оскільки в процесі перевірки моделі система, що розглядається, повинна бути описана за допомогою формальної мови, нам потрібен відповідний для реалізації запропонованого методу. Система перетворення графів (GTS) – це формальна мова для моделювання динамічних структур і поведінки різних систем. Крім того, GTS використовується в багатьох видах діяльності з розробки програмного забезпечення, таких як проектування архітектурних стилів, вдосконалення, мета моделювання [4]та трансформація моделі. Отже, у цій статті GTS розглядається як тестовий стенд для реалізації нашого підходу. Щоб оцінити ефективність запропонованого підходу порівняно отримані результати на різних бенчмарках з існуючими сучасними методиками.

1.1.2 Моделі та компоненти інформаційних систем

Шість компонентів, які повинні об’єднатися, щоб створити інформаційну систему, це:[27]

Апаратне забезпечення: термін апаратне забезпечення стосується машин і обладнання. У сучасній інформаційній системі до цієї категорії відноситься сам комп'ютер і все його допоміжне обладнання. Допоміжне обладнання включає пристрої введення та виведення, пристрої зберігання та пристрої зв'язку. У докомп'ютерних інформаційних системах апаратне забезпечення може включати книги та чорнило.

Програмне забезпечення: термін програмне забезпечення відноситься до комп'ютерних програм і посібників (якщо є), які їх підтримують. Комп'ютерні програми – це машинозчитувані інструкції, які керують схемами апаратних частин системи, щоб вони функціонували таким чином, щоб отримати корисну інформацію з даних. Програми зазвичай зберігаються на якомусь носії введення/виведення, часто на диску або стрічці. «Програмне забезпечення» для докомп'ютерних інформаційних систем включало те, як обладнання було підготовлено до використання (наприклад, заголовки стовпців у книзі бухгалтерської книги) та інструкції щодо їх використання (довідник для карткового каталогу).

Дані: дані – це факти, які використовуються системами для отримання корисної інформації. У сучасних інформаційних системах дані, як правило, зберігаються в машиночитаній формі на диску або стрічці, поки вони не знадобляться комп'ютеру. У докомп'ютерних інформаційних системах дані, як правило, зберігаються у формі, зрозумілій людині.

Процедури: процедури – це політики, які керують роботою інформаційної системи. «Процедури для людей – це те ж саме, що програмне забезпечення для апаратних засобів» – поширена аналогія, яка використовується для ілюстрації ролі процедур у системі.

Люди: кожна система потребує людей, якщо вона хоче бути корисною. Часто елементом системи, якого найбільше не враховують, є люди, ймовірно, компонент, який найбільше впливає на успіх або провал інформаційних систем. Це стосується «не тільки користувачів, але й тих, хто керує та

обслуговує комп’ютери, тих, хто підтримує дані, і тих, хто підтримує мережу комп’ютерів».[28]

Інтернет: Інтернет – це поєднання даних і людей. (Хоча цей компонент не є обов’язковим для функціонування.)

Дані – це міст між обладнанням і людьми. Це означає, що дані, які ми збираємо, є лише даними, поки ми не залучимо людей. У цей момент дані тепер є інформацією.

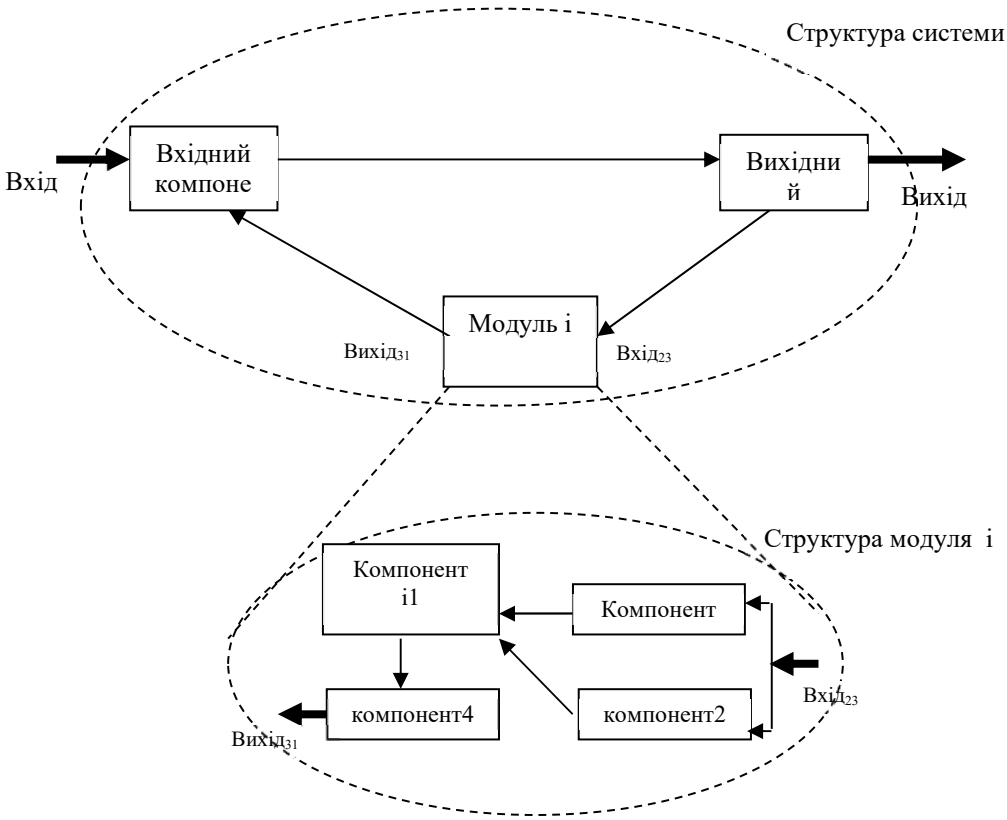

Рисунок 1.1 – Багаторівнева структура системи

Для нової системи концептуальна модель (така ж бізнес-концептуальна модель) є важливою відправною точкою для повного розуміння бізнес-вимог. Для проектів, які вимагають переходу від попередньої розробки до новішої технології, існуюча концептуальна модель допоможе розробнику зрозуміти попередні вимоги користувачів до старого джерела даних. Незалежно від того, чи йдеться про новий проект розробки чи про перехід до нової технології,

концептуальна модель забезпечує семантичне розуміння бізнес-логіки та правил, виражених у концептуальній моделі як зв’язки. Діаграма нижче є концептуальною моделлю, заснованою на подорожі залізницею. Поїзд пов’язано з локомотивом у відношенні M:N (багато до багатьох), а потяг додатково пов’язано з багатьма вагонами у відношенні один (поїзд) до багатьох (вагон). Ця концептуальна модель виражає бізнес-вимоги та правила, які охоплюють (а) те, що поїзд пов’язаний з (або багатьма) локомотивами та (б) поїзд має один або багато вагонів, як приклад двох бізнес-правил. Дивіться рисунок 2.1 нижче.

Концептуальні моделі представляють дані, зібрані з бізнес-вимог, визначають ключові бізнес-об’єкти (наприклад, локомотив або потяг у прикладі концептуальної моделі) і зв’язок між бізнес-об’єктами, які встановлюють бізнес-правила. Зв’язки відносин, відображені в концептуальній моделі, знову ж таки, характеризуються потребами бізнесу. У концептуальній моделі логіка групується навколо основних елементів або об’єктів, отриманих на основі аналізу вимог, і розміщується в інструменті моделювання, який підтримує використання основних сутностей, які представляють намір аналізу вимог. Суб’єкти та зв’язки далі розвиваються навколо потреб бізнесу. Розробка концептуальної моделі вимагає введення користувача (або введення бізнес-аналітика), щоб розробник бази даних міг повністю зрозуміти та реалізувати технічні аспекти бізнес-вимог до нової бази даних або переміщеної бази даних.

Загалом концептуальну модель можна розглядати як карту бізнес-концепцій та їхніх зв’язків для бази даних. Концептуальна модель надає ділову перспективу того, як користувач використовує інформацію. У концептуальній моделі некритичні технічні деталі пригнічуються, щоб дозволити акцентувати увагу на бізнес-правилах і об’єктах користувача (наприклад, сутності), які мають відношення до бізнес-перспективи бази даних.

1.1.3 Процесна структура системи і тупикової

Взаємоблокування – це ситуація, коли набір процесів заблоковано, тому що кожен процес утримує ресурс і очікує на інший ресурс, отриманий іншим процесом.

Розглянемо приклад, коли два поїзди йдуть назустріч один одному по одній колії, і є лише одна колія, жоден із поїздів не може рухатися, коли вони стоять один перед одним. Подібна ситуація виникає в операційних системах, коли є два або більше процесів, які зберігають одні ресурси та чекають ресурсів, які зберігаються іншими.

Взаємоблокування виникає, якщо обидва процеси переходять до другого запиту. Взаємоблокування може виникнути, якщо одночасно виконуються наступні чотири умови (необхідні умови).

Взаємне виключення: два або більше ресурсів не можна спільно використовувати (одночасно можна використовувати лише один процес)

Утримувати та чекати: процес утримує принаймні один ресурс і очікує на ресурси.

Без преференції: ресурс не можна взяти з процесу, якщо процес не звільнить ресурс.

Циклове очікування: набір процесів, які очікують один одного в циклічній формі.

Методи обробки тупикових блокувань – існує чотири підходи до вирішення тупикових блокувань:

- попередження тупикових ситуацій;
- уникнення тупикової ситуації (алгоритм банкіра);
- виявлення та відновлення взаємоблокувань;
- тупикова невігласство (метод страуса).

Стратегія запобігання тупиковим блокуванням полягає в проектуванні системи таким чином, щоб виключити можливість тупикових блокувань. Непрямі методи запобігають виникненню однієї з трьох необхідних умов

взаємоблокування, тобто взаємне виключення, відсутність випередження, утримання та очікування. Прямий метод запобігає виникненню циклічного очікування.

Методи запобігання – взаємне виключення – підтримуються ОС. Затримувати та чекати – умові можна запобігти, вимагаючи, щоб процес запитував усі необхідні ресурси одночасно, і блокуючи процес, доки всі його запити не будуть надані одночасно. Але ця профілактика не дає хороших результатів, оскільки: потрібен тривалий час очікування, неефективне використання виділеного ресурсу, Процес може заздалегідь не знати про всі необхідні ресурси.

Алгоритм уникнення тупикових ситуацій працює шляхом активного пошуку потенційних тупикових ситуацій до їх виникнення. Це робиться шляхом відстеження використання ресурсів кожним процесом і виявлення конфліктів, які потенційно можуть призвести до тупикової ситуації. Якщо виявлено потенційну тупикову блокування, алгоритм вживе заходів для вирішення конфлікту, наприклад відкочує один із процесів або запобіжний розподіл ресурсів для інших процесів. Алгоритм уникнення взаємоблокування розроблено для мінімізації ймовірності виникнення взаємоблокування, хоча він не може гарантувати, що взаємоблокування ніколи не відбудеться. Цей підхід допускає три необхідні умови блокування, але робить розумний вибір, щоб гарантувати, що точка блокування ніколи не буде досягнута. Це забезпечує більше паралелізму, ніж виявлення уникнення. Рішення приймається динамічно, чи призведе поточний запит на виділення ресурсу, якщо він буде наданий, потенційно призвести до тупикової блокування. Це вимагає знання майбутніх запитів процесу. Два способи уникнути тупикової ситуації:

- відмова в ініціації процесу;
- відмова у виділенні ресурсу.

Переваги методів уникнення тупикових ситуацій:

- немає необхідності випереджати та відкочувати процеси;

- менш обмеження, ніж запобігання взаємоблокуванням.

Недоліки:

- майбутні потреби в ресурсах повинні бути відомі заздалегідь;
- процеси можуть бути заблоковані на тривалий час;
- існує фіксована кількість ресурсів для розподілу.

Алгоритм Банкіра базується на концепції графіків розподілу ресурсів.

Граф розподілу ресурсів – це орієнтований граф, де кожен вузол представляє процес, а кожне ребро – ресурс. Стан системи представлено поточним розподілом ресурсів між процесами. Наприклад, якщо в системі є три процеси, кожен з яких використовує два ресурси, графік розподілу ресурсів виглядатиме так:

Процеси A, B і C будуть вузлами, а ресурси, які вони використовують, – ребрами, що їх з'єднують. Алгоритм працює, аналізуючи стан системи та визначаючи, чи перебуває вона в безпечному стані чи під загрозою входження в тупик.

Щоб визначити, чи перебуває система в безпечному стані, Алгоритм Банкіра використовує дві матриці: доступну матрицю та матрицю потреби. Доступна матриця містить кількість кожного ресурсу, доступного на даний момент. Матриця потреб містить кількість кожного ресурсу, необхідного для кожного процесу.

Виявлення взаємоблокувань використовується за допомогою алгоритму, який відстежує циклічне очікування та вбиває один або кілька процесів, щоб усунути взаємоблокування. Стан системи періодично перевіряється, щоб визначити, чи набір процесів заблоковано. Взаємоблокування вирішується шляхом переривання та перезапуску процесу, відмовляючись від усіх ресурсів, які утримував процес. Ця техніка не обмежує доступ до ресурсів і не обмежує дії процесу. Запитані ресурси надаються процесам, коли це можливо. Це ніколи не затримує початок процесу та полегшує обробку онлайн. Недоліком є властиві втрати на випередження.

У методі ігнорування взаємоблокування ОС діє так, ніби взаємоблокування ніколи не виникає, і повністю ігнорує його, навіть якщо взаємоблокування виникає. Цей метод застосовується лише в тому випадку, якщо взаємоблокування виникає дуже рідко. Алгоритм дуже простий. У ньому сказано: «якщо трапиться взаємоблокування, просто перезавантажте систему та дійте так, ніби тупикової блокування ніколи не було». Ось чому алгоритм називається алгоритмом Страуса.

Переваги:

- алгоритм страуса відносно простий у реалізації та ефективний у більшості випадків;

- це допомагає уникнути тупикової ситуації, ігноруючи наявність тупикових блокувань.

Недоліки:

- алгоритм страуса не надає жодної інформації про безвихідну ситуацію;

- це може призвести до зниження продуктивності системи, оскільки система може бути заблокована протягом тривалого часу;

- це може призвести до витоку ресурсів, оскільки ресурси не звільняються, коли система заблокована через взаємоблокування.

1.2 Актуальність проблеми

Над проблемою тупиків ведеться робота протягом багатьох років. Тим не менш, оптимальний шлях її розв’язання так поки що не виявлено. Спроби запобігти тупика накладають обмеження на процеси системи, якщо вона працює в паралельному режимі. Таким чином, подібні заходи мають сенс у системах, де ймовірність тупика висока або його наслідки мають вирішальне значення, наприклад, у системах реального часу.

Розглянемо, як це питання вирішується у операційних системах. Сучасні ОС використовують метод "страуса". Такий метод означає повністю

ігнорування проблеми. Це має сенс тому, що ймовірність тупика невелика. Крім того, наслідки тупика не є критичними, і перезапуск системи може вирішити цю проблему. Таким чином, більшість ОС, включаючи Windows та NIX -системи, ігнорують цю проблему. Розробники стверджують, що більшість користувачів воліли б періодично стикатися з тупиком, ніж миритися з ситуацією, коли тільки один процес або тільки один файл допускається до використання.

Однак, незважаючи на відсутність інструментів для пошуку та розв’язання тупиків, ОС мають механізми синхронізації, такі як семафори, критичні секції та м'ютекси. Використання таких механізмів дозволяє організувати спільне використання ресурсів та взаємодію між процесами. Тим не менш, механізми синхронізації не виключають тупика в системі. Наприклад, може виникнути тупик, коли критична секція використовується неправильно.

На відміну від ОС, тупик може призвести до негативних наслідків у розподілених базах даних. Всі розраховані на багато користувачів бази даних мають такі загальні проблеми оновлення та видалення даних [2].

Причиною таких проблем є те, що загальні дані можуть змінюватись різними користувачами. Проблема видалення даних полягає в наступному: один із користувачів БД намагається видалити запис, який використовується в цей момент іншим користувачем. Проблема оновлення ж полягає в тому, що двоє користувачів БД намагаються змінити один і той самий запис.

Для розв’язання цих проблем у СУБД використовуються транзакції. Транзакція – це група послідовних операцій із базою даних, що є логічну одиницю роботи з даними. p align="justify"> Робота транзакції повинна бути завершена в цілому, а якщо в будь-якої частини її не вдається виконання, то вся транзакція не завершується [5]. Щоразу, коли транзакція відбувається, відповідний запис у БД блокується, й у разі лише дана транзакція може його змінити. Припустимо, що транзакція A заблокувала запис a і намагається заблокувати запис b , але A не буде успішно завершено, оскільки цей запис був

заблокований транзакцією B. Таким чином, транзакція A зупиняється і чекає на звільнення запису b.

У той же час транзакція B намагається блокувати запис a. Але запис уже заблоковано, і транзакція B зупиняється. Таким чином, виникає тупикова ситуація.

На відміну від операційних систем, тупик у БД може призвести до втрати або спотворення інформації; тому в сучасних системах управління базами даних існують механізми для пошуку та розв’язання тупикових ситуацій.

Наприклад, розглянемо організацію такого механізму в MS SQL Server [3]. СУБД використовує граф очікування (англ, wait - for graph) для пошуку тупика, де вузли являють собою процеси, а дуги відношення. Наприклад, вузол T_i -> T_j утворюється, якщо T_i чекає, коли T_j звільнить об'єкт . Після кожного блокуючого запиту будується граф. Диспетчер блокування активується через проміжок часу, заданий таймером. Якщо диспетчер виявляє транзакцію, яка перебуває у режимі очікування дуже тривалий час, він ініціює процес пошуку циклів у графі. Якщо тупик виявлено, відбувається відкат однієї транзакції . Жертва визначається за обсягом виконаної роботи, що визначається за кількістю записів у журналі транзакцій.

Таким чином, немає найкращого способу боротьби з тупиками. Складність полягає в необхідності обмежувати процеси . Тож доводиться вибирати між правильністю та функціональністю. Це питання викликає багато дискусій про те, що є головнішим. Однак важко знайти оптимальне рішення.

1.3 Постановка мети та задач роботи

На даний момент найбільш поширеним є метод страуса, а також виявлення тупика і відновлення системи під час її роботи. Ці підходи мають деякі обмеження та не мають єдиних критеріїв для розв’язання тупикової проблеми. Таким чином, через відсутність ефективного розв’язання це завдання, як і раніше, важливе на сьогоднішній день.

Мета кваліфікаційної роботи досягається послідовним вирішенням наступних задач, а саме треба:

- аналіз формальних засобів для опису інформаційної систем;
- аналіз формальних засобів опису проблеми пошуку тупиків;
- аналіз використання процесних моделей для опису складних систем;
- оптимізація моделі структури системи для підвищення ефективності методів усунення тупиків;
- оцінка ефективності методів оптимізації.

2 МАТЕМАТИЧНІ СХЕМИ ДЛЯ ОПИСУ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Ієрархічні рівні представлення інформаційної системи

Основним методом проектування складних систем є блочно-ієрархічний метод [17], за якого складний об’єкт розбивається на скінченну кількість взаємопов’язаних підсистем, а саме блоків. Якщо підсистеми все ще складні, то до кожної з них застосовується метод декомпозиції, і процедура розбиття триває до тих пір, поки підсистеми не будуть визнані досить простими і зручними для прямого математичного опису. Підсистеми, які не підлягають подальшому розчленуванню, будемо називати елементами складної системи.

Декомпозиція системи повинна починатися з обраного рівня абстракції моделі, який визначається трьома факторами: цілями моделювання; обсяг апріорної інформації про систему; вимоги до точності та достовірності результатів моделювання.

Прикладом декомпозиції ієрархічної структури в CS є сучасний мікропроцесор, який містить готівкову пам'ять, контролери, ядро процесора і магістраль, до якої підключені чіпсети. Ядро процесора можна розкласти на більш прості компоненти: модулі з конвеєрною обробкою, пристрій управління, який можна розкласти і на більш прості елементи, такі як логічне «і», інвертори, тригери і т.д.

Таким чином, в результаті застосування методу декомпозиції утворюється багаторівнева ієрархічна система блоків. Таку систему зручно зобразити у вигляді дерева (рисунок 2.1). Очевидно, що рівні цього дерева відповідають різним рівням ієрархії. Листя дерева визначає поведінку елементів найнижчого рівня.

Ми визначаємо «блок» як елемент системи, який виконує певні функції і не підлягає декомпозиції на даному рівні абстракції.

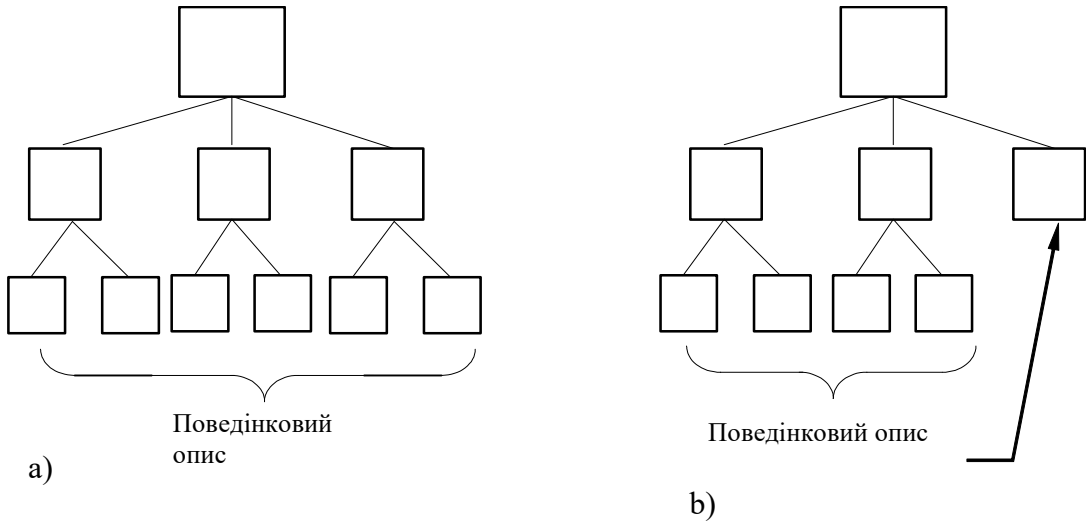

Рисунок 2.1 – Багаторівнева ієрархічна система блоків, зображена у вигляді: а) повного дерева; б) неповного дерева

Кількість ієрархічних рівнів завжди обмежена. Підвищення рівня деталізації (зниження рівня абстракції) систем повинно бути здійснено до такого ступеня, щоб залежності вводу-виводу були відомі або могли бути отримані для кожного блоку. Підвищення рівня деталізації дозволяє вивести більш точну модель, але ускладнює процес моделювання та збільшує витрати часу на його проведення.

Метод декомпозиції породжує серйозні проблеми при створенні системи автоматичного проектування:

- визначення ієрархічних рівнів та засад для них;
- перехід від однієї основи до іншої;
- розробка формального підходу до опису аналізованої системи.

Метод декомпозиції базується на двох способах представлення компонентів системи: структурному та поведінковому.

2.2 Структурний та поведінковий підходи до опису складних систем

Структурний підхід описує елемент системи як сукупність взаємопов’язаних елементів нижчого рівня [27]. Структурна форма ієрархії

проекту означає декомпозицію або розщеплення проекту. Після декомпозиції модель системи будь-якого рівня будується як набір взаємопов’язаних елементів, визначених для даного рівня абстракції. Будь-який структурний компонент повинен відповідати зазначеному елементу в системі або якійсь частині елемента.

Поведінковий спосіб забезпечує опис елемента системи за вхідними/вихідними залежностями за допомогою деякої процедури [27]. І цей опис визначається якоюсь власною процедурою і не описується за допомогою інших елементів. Тому для опису елементів-листок дерева проекту використовується поведінкова модель. Оскільки поведінкова модель деякого проекту може існувати на будь-якому рівні, різні частини проекту можуть мати поведінкові описи на різних рівнях. На рисунку 2.1a показано «повне» дерево проекту (де весь опис поведінки формується на одному рівні). На рисунку 2.1b проект показано у формі неповного дерева (де показані описи поведінки, що стосуються різних рівнів). Подібна ситуація виникає тому, що часто розробнику бажано побудувати і проаналізувати взаємозв’язки між компонентами системи ще до завершення проектування. Ефективність моделювання підвищується при використанні цього підходу розробником. Цей факт є додатковою перевагою поведінкового підходу.

2.3 Математичні моделі елементів складних систем

Як було показано в попередньому розділі, складна система являє собою багаторівневу конструкцію, що складається з кластерів елементів.

Як зазначалося раніше, представлення модельованого об'єкта як багаторівневої конструкції елементів відноситься до декомпозиції (структуралізму) об'єкта. Основний висновок, який випливає з принципу декомпозиції, такий: «математична модель складної системи складається з математичних моделей елементів і математичної моделі взаємозв’язків між елементами».

У цьому параграфі ми приділимо увагу математичним моделям елементів складних систем. Очевидно, що характер математичної моделі будь-якого об'єкта матеріального світу істотно залежить від його властивостей і особливостей функціонування. Однак не менш важливими є цілі моделі, тобто дослідження, які передбачається проводити на цій моделі. Тому процес функціонування одного і того ж об'єкта можна описати по-різному, в залежності від завдань моделювання.

Серйозну допомогу при побудові математичних моделей може надати володіння типовими математичними схемами, які широко використовуються в моделюванні.

Вихідною інформацією для побудови математичних моделей систем є дані про призначення та умови роботи аналізованої системи S. Ця інформація визначає основну мету моделювання системи S і дозволяє сформулювати вимоги до математичної моделі М.

У практиці моделювання об’єктів у сфері системного аналізу на етапі початкового дослідження системи раціонально використовувати типові математичні схеми: диференціальні рівняння, кінцеві та ймовірнісні автомати, системи масового обслуговування та ін.

Математичні схеми мають перевагу в простоті порівняно з узагальненими моделями, але їх не завжди можна застосувати. Якщо випадкові фактори не враховуються, модель стає детермінованою. Для зображення таких систем, що функціонують у безперервному часі, можна використовувати диференціальні, інтегральні, інтегро-диференціальні та інші рівняння, а для зображення систем, що функціонують у дискретному часі, – кінцеві автомати. В якості стохастичних моделей (при врахуванні випадкових факторів) для представлення систем з дискретним часом можна використовувати ймовірнісні автомати, а для представлення системи з безперервним часом – системи черги тощо.

Природно, що не всі процеси, що відбуваються у великих інформаційних системах, можна представити математичними схемами. У деяких випадках більше підходять агрегатні моделі.

Агреговані моделі (системи) дозволяють описувати більш широке коло об'єктів і враховують системні особливості цих об'єктів. При сукупному описі складний об'єкт (система) розчленовується на кінцеве число частин (підсистем), зберігаючи зв’язки між частинами.

Таким чином, можна сформулювати такі основні підходи:

- безперервно визначені (наприклад, диференціальні рівняння);
- дискретно-визначені (кінцеві автомати);
- дискретно-стохастичні (імовірнісні автомати);
- безперервно-стохастичні (системи масового обслуговування);
- агрегатний.

Безперервно визначені моделі (D-ланцюг) можна описати диференціальними рівняннями. Особливості безперервно детермінованого підходу розглянемо на прикладі використання диференціальних рівнянь як математичних моделей. Зазвичай у таких математичних моделях час t є незалежною змінною.

Таким чином, використання D-схем дозволяє формалізувати процес функціонування неперервно-детермінованих систем S та оцінити їх основні характеристики, застосовуючи аналітичний або імітаційний підхід, який реалізується у вигляді певної мови для моделювання неперервних систем або з використанням аналогових та гібридних засобів обчислювальної техніки. .

Особливості дискретно-детермінованого підходу розглянемо на прикладі використання теорії автоматів як математичного інструменту. За допомогою цієї теорії система представлена автоматами, які передають дискретну інформацію і змінюють свою сутність лише протягом допустимих моментів часу. Поняття автоматизації змінюється в залежності від характеру конкретно досліджуваних систем і від прийнятого рівня абстракції.

Автоматику можна представити як деякий об'єкт (чорний ящик), на який надходять вхідні сигнали і який може мати певну внутрішню приналежність. Автомати є скінченними автоматами, якщо множина внутрішніх і вхідних сигналів (отже, і множина вихідних сигналів) остаточно задана.

Скінчений автомат – це математична схема (F-схема), яка описується шістьма елементами таким чином:

$$F = < Z, X, Y, \varphi, \psi, z_0 >, \qquad (2.1)$$

де Z – остаточний набір внутрішніх ознак (внутрішній алфавіт або алфавіт умов), X – кінцевий набір вхідних сигналів (вхідний алфавіт), Y – кінцевий набір вихідних сигналів (вихідний алфавіт), φ (z, x) – функція переходу, ψ (z, x) – функція виходу та z_0 – початкова умова, $z_0 \in Z$. Автомати, задані F-ланцюгом, працюють у дискретному часі.

Цей підхід не підходить для опису процесів прийняття рішень, процесів у динамічних системах з наявністю перехідних процесів і стохастичних елементів.

Дискретно-стохастичні моделі (P-ланцюг). Розбиття часу в цьому підході залишається подібним до розглянутої вище кінцевої автоматизації. У загальному випадку ймовірнісні автомати можна визначити як дискретний перетворювач інформації. Цей автомат має внутрішню пам'ять. Функціонування цього автомата на кожному кроці залежить тільки від стану внутрішньої пам'яті і може бути описано статистично.

Безперервно-стохастичні моделі (Q-ланцюг). Особливості безперервно-стохастичного підходу розглянемо на прикладі використання систем масового обслуговування як типових математичних схем, які назвемо Q-схемами. Системи масового обслуговування являють собою клас математичних схем, розроблених в теорії систем масового обслуговування та різних додатках для формалізації процесів обслуговування.

Процеси функціонування економічних, виробничих, технічних та інших систем можна представити Q-схемою. Наприклад: потоки поставок продукції на якесь підприємство, потоки деталей і комплектуючих виробів на складальному конвеєрі заводу, обробка інформації комп'ютером і т.д. Таким чином, особливістю таких систем є стохастичний характер процесу роботи.

У цьому параграфі ми обговорили всі типові математичні схеми для представлення КС, крім Aggregate або А-схеми. Aggregate (А-схема) має дуже сильний математичний опис і порівняно з розглянутими вище схемами його можна використовувати для опису будь-яких об’єктів, що аналізуються. Іншими словами, ця схема має більш високий рівень абстракції. Давайте розглянемо цю схему точніше.

2.4 Агрегатні моделі (А-ланцюг)

Найпопулярнішим загальним підходом для формального опису системи є той, що запропонований [36]. Такий підхід дозволяє описати поведінку неперервних і дискретних, детермінованих і стохастичних систем. Таким чином, даний підхід є універсальним порівняно з розглянутою вище схемою і ґрунтується на концепції агрегатної системи, що представляє собою формальну універсальну схему, яка називається А-ланцюгом.

Аналіз існуючих засобів моделювання систем і задач, які розв’язуються за допомогою методу комп’ютерного моделювання, приводить до висновку, що комплексне розв’язання задач, що виникають під час проектування та моделювання моделі, можливе лише у випадку, коли моделювана система є побудована на основі універсальної формальної математичної схеми, тобто А-схеми. Така схема повинна задовольняти декільком вимогам: бути адекватним математичним описом вихідного об'єкта, бути основою для побудови алгоритмів і програм для моделі М, дозволяти проводити аналітичні дослідження простим способом.

Отримані вимоги досить суперечливі. Тим не менш, в рамках узагальненого підходу на основі А-схем можна знайти певний компроміс між ними.

При сукупному описі складний об'єкт (система) розбивається на кінцеву сукупність підсистем, зберігаючи зв’язки, що забезпечують їх взаємозв’язки. У результаті такої декомпозиції складна система представляється як багаторівнева конструкція з взаємопов’язаних елементів, об’єднаних у підсистеми різних рівнів.

Елемент А-ланцюга – агрегат. Зв’язок між агрегатами (всередині системи S і з середовищем E) здійснюється за допомогою оператора інтерфейсу R. Сам агрегат можна розглядати як А-ланцюг, тобто розбити на елементи (агрегати) наступного рівня: .

Будь-який агрегат характеризується такими множинами: час T, вхідний X і вихідний Y сигнали, стани Z протягом кожного часу t. Стан агрегату на рівні часу $t \in T$ позначається як $z(t) \in Z$, а вхідний і вихідний сигнали як $x(t) \in X$ і $y(t) \in Y$ відповідно.

Припустимо, що перехід агрегату зі стану z(t1) у стан $z(t2) \neq z(t1)$ відбувається за малий проміжок часу, тобто має місце стрибок δz. Переходи агрегату зі стану $z(t_1)$ в $z(t_2)$ визначаються власними (внутрішніми) параметрами агрегату $h(t) \in H$ і вхідними сигналами $x(t) \in X$.

У початковий момент часу (t_0) стан z має значення $z_\circ = z(t_0)$. Якщо процес функціонування агрегату (при надходженні вхідного сигналу xn) описувати ймовірнісним оператором V, то можна визначити стан агрегату в момент надходження в агрегат вхідного сигналу x_n за формулою:

$$z(t_n + 0) = V[t_n, z(t_n), x_n] \quad . \qquad (2.2)$$

Припустимо підінтервал часу $t_1 < t \leq t_2$ як $(t_1, t_2]$, а підінтервал $t_1 \leq t < t_2$ як $[t_1, t_2)$. Якщо проміжок часу (t_n, t_{n+1}) не містить жодного моменту прийому

сигналів, то для $t \in (t_n, t_{n+1})$ стан агрегату визначається ймовірнісним оператором U за виразом

$$z(t) = U[t, t_n, z(t_n + 0)]. \quad (2.3)$$

Сукупність імовірнісних операторів V і U розглядається як оператор переходів агрегату в новий стан. Таким чином, процес функціонування агрегату складається зі стрибків станів δz у момент надходження вхідних сигналів x (оператор V) та зміни станів між моментами t_n і t_{n+1} (оператор U). Оператор U не має жодних обмежень. Моменти стрибків δz називають особливими моментами часу t_δ, а стани $z(t_\delta)$ – особливими станами А-контуру. Для опису стрибків станів z в особливі моменти часу t використовується ймовірнісний оператор W, що представляє окремий випадок оператора U, тобто.

$$z(t_\delta + 0) = W[t_\delta, z(t_\delta)], \quad (2.4)$$

Підмножина Z(Y) є підмножиною множини станів Z. Якщо досягає Z(Y), то цей стан є моментом генерації вихідного сигналу, який визначається оператором виходів

$$y = G[t_\delta, z(t_\delta)], \quad (2.5)$$

Таким чином, під Агрегатом будемо розуміти будь-який об’єкт, який визначається впорядкованою множиною розглянутих множин T, X, Y, Z, Z(Y), H та ймовірнісних операторів V, U, W, G.

Послідовність вхідних сигналів, відсортованих за способом надходження в А-ланцюг, будемо називати вхідним повідомленням або x-

повідомленням. Послідовність вихідних сигналів, упорядковану за часом генерації, назвемо вихідним повідомленням або у-повідомленням.

Існує клас складних систем, які через їх складність не можуть бути формалізовані як математичні схеми єдиної сукупності, тому їх можна формалізувати як деяку конструкцію з окремих агрегатів An, які називаються агрегатною системою або А-схемою. Для опису деякої реальної системи S за допомогою А-ланцюга необхідно мати опис як окремих агрегатів An, так і зв'язків між ними.

Кожен агрегат An А-ланцюга має вхідні контакти, на які надходить набір елементарних сигналів xi (t), що одночасно виникають на вході елемента, і вихідні контакти, з яких знімається набір елементарних сигналів yi (t). Таким чином, кожен агрегат An А-ланцюга має In вхідні та Jn вихідні контакти.

Формальне визначення А-ланцюга, наведене в цьому параграфі, ми будемо використовувати в наступному розділі для формалізації процесів.

2.5 Модель взаємодії елементів

Взаємодія елементів під час функціонування складної системи розглядається як результат впливу кожного елемента на інші елементи [36]. Вплив назвемо сигналом або подією. Таким чином, взаємодія елементів представлена обміном сигналами.

Сигнали передаються по каналах між елементами КС. Початок каналу - це вихід (цільовий порт) елемента, що видає сигнал, а кінець - вхід (вхідний порт) елемента, що приймає сигнал. Канал, який передає сигнали миттєво і без спотворень (тобто характеристики сигналу на початку і в кінці каналу збігаються в будь-який момент), є ідеальним каналом.

Зазвичай фізичні канали не є ідеальними і їх слід розглядати як самостійні елементи системи, функціонування яких зводиться до відповідних затримок і спотворень сигналів. Таким чином, повністю формалізована система має лише ідеальні канали.

При побудові математичної моделі КС необхідно враховувати взаємодію з навколишнім середовищем. Середовище розглядається як деяка сукупність об'єктів, які впливають на елементи КС, а також сприймають впливи, що виходять від елементів КС. Тому взаємодія з навколишнім середовищем зводиться до механізму обміну сигналами між елементами КС і з об'єктами навколишнього середовища. Цей механізм повністю схожий з механізмом взаємодії між елементами всередині КС.

Механізм обміну сигналами включає такі складові:

- формування вихідного сигналу;
- визначення адреси передачі для кожної характеристики вихідного сигналу;
- передача сигналів по каналах і збір вхідних сигналів для елементів, що приймають сигнали;
- реакція елемента, що приймає сигнал.

Перший і четвертий компоненти не розглядаються в рамках моделі взаємодії. Формування вихідних сигналів і реакція на вхідні сигнали включається безпосередньо в процес побудови математичних моделей елементів КС. Без нього функціональні переходи (або оператори) і виходи не можуть бути правильно обрані, і навіть набори вхідних і вихідних сигналів не можуть бути визначені для них. Таким чином, побудова математичних моделей поведінки елементів і взаємодії елементів є єдиною задачею. Розглядати ці проблеми окремо одну від одної практично неможливо.

Третя складова механізму обміну сигналами пов’язана із затримками і спотвореннями сигналів через неідеальні канали. Розгляд фізичного каналу зв’язку як самостійного елемента складної системи та побудова його математичної моделі як динамічної системи відповідного типу базується на результатах і висновках спеціальних наук, пов’язаних з теорією передачі інформації. Дана система повністю формалізована, отже, система має лише ідеальні канали зв’язку.

Таким чином, необхідно розглянути другу складову механізму обміну сигналами в КС – адресацію вихідних сигналів.

2.6 Висновки

У цій главі були розглянуті базові питання, що стосуються уявлень і описів КС. Ми обговорили метод декомпозиції вихідної КС, щоб зробити аналізовану систему більш зрозумілою та придатною для представлення її за допомогою типових математичних схем: диференціальних рівнянь, кінцевих та імовірнісних автоматів, систем масового обслуговування та агрегатів або А-схеми. Агрегат (А-схема) обговорювався точніше, оскільки А-ланцюг дуже схожий на іншу схему високого рівня абстракції, яка називається процесом. При цьому формальний опис агрегату можна використовувати для розробки формального опису процесу. Після цього ми обговорили модель взаємодії елементів і схему інтерфейсу між елементами. Ці дослідження будуть використані в наступних розділах для роботи над розробкою матеріалу стосовно КС, способів вибору алгоритмів та методів для вирішення проблем в ситуаціях взаємоблокування та моментів виявлення тупиків на етапі проектування чи на більш пізніших етапах розробки різноманітних інформаційних систем.

3 МАТЕМАТИЧНІ СХЕМИ ДЛЯ ОПИСУ СКЛАДНИХ СИСТЕМ НА БАЗІ ПРОЦЕСІВ

3.1 Визначення процесу

На даний момент в літературі немає усталеного визначення «Процесу» як математичної абстракції, що використовується для представлення різних моделей (включаючи паралельні). Тим не менш, мови, які використовуються для опису таких моделей, часто включають механізми, які дозволяють:

- внести в процес механізм вибору різних варіантів його поведінки (альтернативний склад);
- скласти процес, в якому набір різноманітних варіантів поведінки здійснюється послідовно (послідовна композиція);
- скласти процес, в якому набір різних варіантів поведінки здійснюється паралельно (паралельна композиція).

Гл. Хоар [15] вводить поняття процесу як деякої сукупності подій, яку він називає алфавітом процесу. Виникнення події переносить процес у відповідний стан. Такий перехід процесу відбувається за алгоритмом.

Дейтель [12] використовує термін «процес» як синонім:

- завдання (набір програмних модулів, для реалізації яких потрібні ресурси обчислювальної системи);
- програма в стадії виконання;
- процедура;
- блок керування процесом в операційній системі.

Таким чином, у цій роботі ми розглядатимемо процес як інформаційну структуру, яка містить щонайменше два компоненти: список подій (тобто алфавіт процесу) та алгоритм, що описує поведінку процесу.

Визначимо процес як об'єкт, який виконує певну діяльність або дії (наприклад, обчислювальні дії) відповідно до алгоритму і вимагає для цього ресурсів із середовища, в якому він виконується.

Поняття процесу в такому трактуванні узагальнює відоме раніше визначення процесу і передбачає такі важливі аспекти: по-перше, це носій даних; по-друге, виконує операції, пов’язані з їх обробкою, і, по-третє, вимагає ресурсів для зберігання та обробки даних. У рамках запропонованої концепції процесу проблему проектування механізмів розподілу та управління ресурсами слід розглядати разом із проблемою управління процесами.

Приклад процесу. Як приклад процесу в складній електронній системі можна розглянути функціонування процесора Pentium (функціонування його конвеєрів). Процесор Pentium має два конвеєри (U і V), які функціонально схожі, але другий (V) порівняно з основним (U) має деякі обмеження [8]. Роботу таких конвеєрів можна представити у вигляді двох паралельних процесів зі складною системою синхронізації та здійснення різноманітних переходів (перевибір команд, обробка команд розгалуження тощо). Поняття процесу є фундаментальним для процесорів, побудованих на нейронних технологіях. У таких процесорах виконання обчислень представлено у вигляді паралельних процесів. Це обумовлено архітектурою таких комп'ютерів.

3.2 Стани процесу

Протягом свого існування процес проходить через ряд особливих дискретних станів. Різні події можуть викликати зміну цих станів. Прийнято розрізняти наступні спеціальні стани процесу: генерація, готовність, активний стан або стан виконання, очікування або блокування, припинення. Розглянемо ці стани докладніше.

Генерація процесу полягає в підготовці умов для його першого виконання. Створення або генерація процесу складається з багатьох операцій, таких як: присвоєння імені процесу; включення цього імені в список імен

процесів, відомих системі; визначення початкового пріоритету процесу; виділення початкових ресурсів процесу тощо. Це пасивний стан процесу, який не конкурує за ресурси (хоча його існування в системі вже пов’язане з наданням йому ресурсів, наприклад, оперативної та/або зовнішньої пам'яті).

Стан готовності означає, що процес не виконується, хоча всі необхідні ресурси для його виконання можуть бути надані на поточному рівні часу.

В активному стані або стані продуктивності процес виконується і може конкурувати за ресурси.

У стані очікування або блокування попередній процес не виконується, оскільки необхідні ресурси не можуть бути надані. Цей процес очікує виникнення певної події, щоб мати можливість продовжити, наприклад: завершення операції введення-виведення.

Звичайне або аварійне завершення виконання процесу відбувається, коли задано завершення стану, після якого йому більше не надаються ресурси.Для завершення процесу викладених вище станів розглянемо приклад конвеєра процесора Pentium. Під час роботи конвеєра наступні стани: простого конвеєра, повторний вибір команди, покрокове виконання завантаженої команди, очищення конвеєра. Якщо інтерпретувати функціонування конвеєра в рамках процесу, то реальні стани можуть відповідати представленим раніше станом процесу, як: простого конвеєра – це стан очікування або блокування; перевибір команди – готовність; покрокове виконання завантаженої команди є умовою виконання (активний стан); очищення конвеєра - закінчення.

Протягом деякого часу процес може перебувати в кожному з характерних станів.

Процес може переходити в інший характерний стан відповідно до настання певної події.

Можливі переходи процесу з одного стану в інший відображаються у вигляді графа потоку станів.

3.3 Формальне визначення процесу

Процес має входи, виходи, внутрішні; кожен процес виконує певний алгоритм, що задає поведінку модельованої функції системи, і характеризується режимом роботи. Процеси моделювання можуть здійснюватися паралельно.

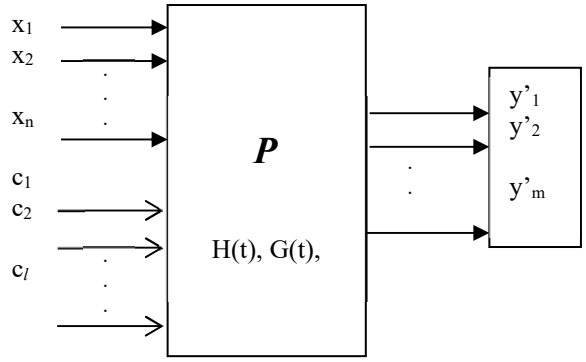

Рисунок 3.1 – Модель процесу

Вхідні дані процесу поділяють на дві групи: керуючі та інформаційні. Інформаційні входи $x=(x_1,x_2,...,x_n)$ приймають значення з алфавіту X і є функцією часу $x(t)$. Керуючі входи приймають значення з алфавіту C і також є функцією часу $c(t)$. Для спрощення опису будемо вважати, що всі сигнали xi і ci є елементарними, що їх не можна представити у вигляді набору каналів (транка). Якщо процес працює зі складними сигналами, вони представлені у вигляді кількох елементарних сигналів.

Розбиття набору вхідних сигналів на дві основні групи виконано відповідно до їх впливу на функціонування системи. Один сигнал просто представляє дані проходження в системі на різних етапах обробки. Ці сигнали є інформацією. Поведінка системи залежить від того, яка інформація проходить по кожному каналу.

З сигналами $c(t)$ спостерігається інша ситуація. Ці сигнали визначають час початку або закінчення виконання деяких функцій процесу. Таким чином, можна сказати, що інформаційні сигнали синхронізуються менеджерами.

Виходи процесу не мають поділу на керуючий та інформаційний. Вихід також є вектором $y' = (y_1', y_2', ..., y_m')$, що представляє функцію часу $y'(t)$.

Кожен вихідний елементарний сигнал є структурою, яка описується виразом:

$$y_i' = (y_i, \Delta t_i, a_i), \tag{3.1}$$

де y_i – наступне значення результату процесу I; сигнали y_i приймають значення з алфавіту Y;

Δt_i – це інтервал часу, через який вихід і значення процесу повинен бути встановлений. $\Delta t \in T, T = [0, \infty)$;

a_i – вектор додаткових атрибутів, що впливають на спосіб моделювання затримки на даній лінії $a_i \in A$.

Також процес оперує локальними (внутрішніми) даними, які представлені змінними в їх традиційному розумінні. В якості локальних даних процесу можуть виступати нутрощі, що описують можливі режими роботи процесу, момент часу, на якому підвішений його внутрішній алгоритм, покажчик поточної виконуваної інструкції алгоритму процесу (ІП) тощо. $z = (z_1, z_2, ..., z_k)$ вектором позначте сутність процесу. Елементи цього вектора приймають значення з алфавіту Z і називаються фазовими координатами. Протягом часу t стани також змінюються, тобто є функцією часу $z(t)$.

Як вже було сказано, в кожен момент часу процес знаходиться в певному робочому режимі $m_i \in M, M = \{active, sleep, ...\}$, де active – активний стан процесу, в якому виконується його алгоритм, sleep - стан очікування і т.д.

В активному стані кожен процес характеризується поведінкою, яка визначається імітаційним алгоритмом. Форма представлення алгоритму може бути будь-якою: операційні вирази, графи потоків процесів, таблиці, словесні описи, програма на мові моделювання та ін.

Розглянемо тепер динаміку певного процесу. На початку моделювання (t0=0) відбувається ініціалізація всіх процесів моделі. Визначається оператором ініціалізації процесу Dinit. Відповідно до цього оператора початкові значення присвоюються всім внутрішнім змінним процесу, покажчик поточної інструкції IP встановлюється на першому операторі алгоритму для кожного процесу, а також усі процеси моделі переходять у режим готовності.

Далі кожен процес моделі починає виконання алгоритму. База часу, розділена на частини. Межами часових рівнів є моменти зміни вхідних або вихідних сигналів. Такі часові рівні називаються особливими моментами. Стани в особливі моменти часу називаються особливими станами. Тоді вся динаміка процесу описується чергуванням особливих станів. Під час виконання алгоритму кожен процес може змінювати значення своїх внутрішньостей, обчислювати нові значення виходів і затримок, які потрібно обробити на кожному з виходів. Таким чином, алгоритм процесу є програмною реалізацією наступних двох груп операторів:

$$z(t_2) = H_1(z(t_1), x(t_1), c(t_1)), \tag{3.2}$$

де t1 – поточне значення часу моделювання;

t2 – наступний рівень часу під час моделювання, значення якого відрізняється від t1 нескінченно малою величиною;

H1 – оператор переходів.

Друга група операторів визначає виходи процесу:

$$y(t_2) = G_1(z(t_1), x(t_1), c(t_1)), \quad (3.3)$$

$$\Delta t(t_2) = G_2(z(t_1), x(t_1), c(t_1)), \quad (3.4)$$

$$a(t_2) = G_3(z(t_1), x(t_1), c(t_1)), \quad (3.5)$$

$$y'(t_2) = G_4(y(t_2), \Delta t(t_2), a(t_2)), \quad (3.6)$$

де оператор G1 обчислює значення сигналу y на виході процесу $y\Delta t \in T, T = [0, \infty)$; оператор G2 обчислює значення інтервалу часу, через який на виході процесу має бути встановлено значення; оператор G3 визначає атрибути, що впливають на спосіб моделювання затримки на даній лінії, оператор G4 створює векторну структуру вихідного сигналу y', відповідно до

$$yi\ ': = yi\ \&\ \Delta t\ \&\ ai, \quad (3.7)$$

де операція конкатенації & використовується для створення вектора yi' з трьох елементарних сигналів.

Метою цього оператора є побудова вектора (yi, ti, ai) і передача його на вихід yi' процесу. Зверніть увагу, що цей оператор лише передає результати функцій G1, G2 і G3 на вихід процесу, але не моделює затримку події.

3.4 Аналіз процесних алгебр

Термін алгебра процесів використовується в різних значеннях. Перш за все, розглянемо слово «процес» в рамках алгебри процесів. Це відноситься до поведінки системи. Система – це все, що демонструє поведінку, зокрема виконання програмної системи, дії машини або навіть дії людини. Поведінка - це сукупність подій або дій, які може виконати система, порядок, у якому вони можуть бути виконані, і, можливо, інші аспекти цього виконання, такі як час або ймовірності. Ми завжди описуємо певні аспекти поведінки, ігноруючи інші аспекти, тому ми розглядаємо абстракцію або ідеалізацію «реальної»

поведінки. Швидше, ми можемо сказати, що у нас є спостереження за поведінкою, а дія є обраною одиницею спостереження. Зазвичай дії вважаються дискретними: події відбуваються в певний момент часу, а різні дії розділені в часі. Ось чому процес іноді також називають системою дискретних подій.

Слово «алгебра» означає, що ми використовуємо алгебраїчний/аксіоматичний підхід, говорячи про поведінку. Тобто використовуємо методи і прийоми універсальної алгебри [14]. Щоб провести порівняння, розглянемо визначення групи в математичній алгебрі.

Група має підпис (G, *, u,-1) із законами або аксіомами:

$$a * (b * c) = (a * b) * c;$$
$$u * a = a = a * u;$$
$$a * a^{-1} = a^{-1} * a = u.$$

Отже, група – це будь-яка математична структура з операторами, що задовольняють аксіомам групи. Іншими словами: група – це будь-яка модель екваціональної теорії груп. Подібним чином можна сказати, що алгебра процесу – це будь-яка математична структура, яка задовольняє аксіомам, наведеним для основних операторів. Процес є елементом алгебри процесу. Використовуючи аксіоми, ми можемо виконувати обчислення з процесами.

Найпростіша модель поведінки – розглядати поведінку як функцію введення/виведення. Значення або вхід надається на початку процесу, а в якийсь момент є (інше) значення як результат або вихід. Ця модель була використана з перевагою як найпростіша модель поведінки комп’ютерної програми в інформатиці з самого початку предмета в середині двадцятого століття. Він відіграв важливу роль у розвитку теорії автоматів (кінцевого стану). У теорії автоматів процес моделюється як автомат. Автомат має ряд станів і ряд переходів, переходячи з одного стану в (інший) стан. Перехід означає виконання (елементарної) дії, основної одиниці поведінки. Крім того,

існує початковий стан (іноді більше одного) і ряд кінцевих станів. Поведінка – це пробіг, тобто шлях від початкового стану до кінцевого. З самого початку важливо, коли вважати два автомати рівними, що виражається поняттям еквівалентності. Для автоматів основним поняттям еквівалентності є еквівалентність мови: поведінка характеризується набором виконання від початкового стану до кінцевого стану. Алгебра, яка дозволяє міркувати за рівняннями про автомати, є алгеброю регулярних виразів.

Пізніше було виявлено, що цієї моделі бракує в кількох ситуаціях. По суті, чого не вистачає, так це поняття взаємодії: під час виконання від початкового стану до кінцевого стану система може взаємодіяти з іншою системою. Це необхідно для опису паралельних або розподілених систем, або так званих реактивних систем. Маючи справу з взаємодіючими системами, ми кажемо, що ми робимо теорію паралельності, тому теорія паралельності – це теорія взаємодіючих, паралельних і/або розподілених систем. Говорячи про алгебру процесів, ми зазвичай розглядаємо це як підхід до теорії паралелізму, тому алгебра процесів зазвичай (але не обов’язково) матиме паралельну композицію як базовий оператор. Таким чином, можна сказати, що алгебра процесів - це вивчення поведінки паралельних або розподілених систем алгебраїчними засобами. Він пропонує засоби для опису або специфікації таких систем, і, отже, має засоби говорити про паралельну композицію. Крім цього, зазвичай також можна говорити про альтернативну композицію (вибір) і послідовну композицію (послідовність). Більше того, ми можемо міркувати про такі системи за допомогою алгебри, тобто рівнянь. За допомогою цього рівняння ми можемо зробити перевірку, тобто ми можемо встановити, що система задовольняє певну властивість.

Щодо основних законів алгебри процесів. Ми можемо перерахувати деякі, які зазвичай називають структурними або статичними законами. Ми починаємо з заданого набору атомарних дій і використовуємо базові оператори, щоб об’єднати їх у більш складні процеси. Як базові оператори ми використовуємо + для позначення альтернативної композиції, • для

позначення послідовної композиції та || що позначає паралельну композицію. Зазвичай існують також нейтральні елементи для деяких або всіх цих операторів, але ми не розглядаємо їх тут. Ось деякі основні закони (+ зв'язування найслабше, • зв'язування найсильніше):

- x + y = y + x (комутативность альтернативної композиції);
- x + (y + z) = (x + y) + z (асоціативність альтернативного складу);
- x + x = x (ідемпотентність альтернативного складу);
- (x + y) • z = x• z + y• z (права розподільність + над •);
- (x• y) • z = x• (y• z) (асоціативність послідовної композиції);
- x || y = y || x (комутативність паралельної композиції);
- (x || y) || z = x || (y || z) (асоціативність паралельного складу).

Отже, ми можемо сказати, що будь-яка математична структура з трьома бінарними операціями, які задовольняють цим 7 законам, є алгеброю процесу. Найчастіше ці структури формулюються в термінах автоматів, у цьому випадку переважно називаються перехідними системами. Це означає, що перехідна система має ряд станів і переходів між ними, початковий стан і ряд кінцевих станів. Досліджуване поняття еквівалентності зазвичай не є мовною еквівалентністю. Чільним серед досліджуваних еквівалентів є поняття бісимуляції. Часто дослідження перехідних систем, способи їх визначення та еквівалентності на них також вважаються частиною алгебри процесу, навіть якщо теорія рівнянь відсутня.

Таким чином, алгебра процесу – це дослідження відповідних екваціональних теорій з їх моделями, тоді як ширша область, яка також включає дослідження перехідних систем і пов'язаних структур, способи їх визначення та еквівалентності на них, буде називатися теорією процесів.

3.4.1 CCS алгебра процесів

Безсумнівно, центральною особою в історії алгебри процесів є Робін Мілнер. Мілнер розробив свою теорію процесів CCS (Calculus of

Communicating Systems) протягом 1973-1980 років, кульмінацією якої стала публікація книги [59] в 1980 році.

Найдавніші публікації, що стосуються семантики паралельної композиції, сформульованої в рамках денотативної семантики, використовуючи так звані трансдуктори. Він розглядає проблеми, викликані незавершеними програмами, з побічними ефектами та недетермінізмом. Він використовує операції * для послідовної композиції, ? для альтернативної опозиції та || для паралельної композиції.

Пізніше Мілнер представив потокові графіки з портами, де іменований порт синхронізується з портом із його співіменем. Операторами є | для паралельної композиції, обмеження та перемаркування. Символ || тепер зарезервовано для обмеженої паралельної композиції. Для цих операторів викладені статичні закони.

У двох наступних статтях [55, 56] представлено більшість відомого нам CCS. Додано префікс динамічних операторів та альтернативну композицію та забезпечено законами. Як модель використовуються дерева синхронізації. Префікс τ виникає як комунікаційний слід (те, що залишається від синхронізації імені та співназви). Парадигма передачі повідомлень запозичена з [57]. Чергування вводиться як спостереження одного спостерігача системи, що спілкується, і встановлюється закон розширення. Послідовна композиція – це не базовий оператор, а похідний, що використовує зв’язок, абстракцію та обмеження.

Стаття [13] разом із Метью Хеннессі формулює основну CCS із еквівалентністю за спостереженнями та сильною еквівалентністю, визначеними індуктивно. Також вводиться так звана логіка Хеннесі-Мілнера, яка забезпечує логічну характеристику еквівалентності процесу. Далі, книга [14] є стандартним довідником з алгебри процесів

Важливим внеском, який було здійснено одразу після появи [59], є формулювання бісимуляції Девіда Парка [60]. Згодом це стало центральним поняттям у теорії процесу.

3.4.2. CSP алгебра процесів

Тоні Хоар зробив дуже важливий внесок у розвиток алгебри процесів. Хоар опублікував впливову статтю [15] як технічний звіт у 1976 році. Важливим кроком є те, що він повністю відмовляється від глобальних змінних і приймає парадигму передачі повідомлень комунікації, таким чином реалізуючи другу зміну парадигми. Мова CSP (Communicating Sequential Processes), описаний у [57], має синхронний зв'язок і є мовою захищених команд. Модель чи семантика не надаються. Ця стаття надихнула Мілнера таким же чином розглядати передачу повідомлень у CCS.

Модель CSP була розроблена в [15]. Це модель, заснована на теорії слідів, тобто на послідовності дій, які може виконувати процес. Пізніше виявилося, що цієї моделі не вистачає, наприклад через те, що взаємоблокування не зберігається. З цієї причини нова модель, заснована на парах відмов, була представлена в [62] для мови, яка тоді називалася TCSP (Theoretical CSP). Пізніше TCSP знову стали називати CSP. Через деякий час було встановлено, що модель відмови є найменш дискримінаційною моделлю, яка зберігає тупикову поведінку. У мові, завдяки наявності двох альтернативних операторів композиції, можна взагалі обійтися без тихого кроку, такого як τ.

3.4.3 ACP алгебра процесів

Ян Бергстра та Ян Віллем Клоп у 1982 році розпочали роботу над питанням Де Баккера щодо того, що можна сказати про розв'язки незахищених рекурсивних рівнянь. У результаті вони написали статтю [63]. У цій статті вперше використовується термін «алгебра процесів». Цитуємо:

Алгебра процесу над набором атомарних дій A є структурою

$$A = \{A,+,\bullet, ||,a_i(i \varepsilon I), \tag{3.8}$$

де A – набір, що містить

A, a_i – постійні символи, що відповідають $a_i \varepsilon A$.

Позначення:+ (об'єднання), (• конкатенація або композиція, опущена в аксіомах), || (ліве злиття) задовольняють для всіх x, y, z ε **A** та a ε A такі аксіоми:

- PA1 x + y = y + x;
- PA2 x + (y + z) = (x + y) + z;
- PA3 x + x = x;
- PA4 (xy)z = x(yz);
- PA5 (x + y)z = xz + yz;
- PA6 (x + y)||z = x||z + y||z;
- PA7 ax||y = a(x||y + y||x);
- PA8 a||y = ay.

Це чітко встановлює алгебру процесу в строгому сенсі. Алгебра процесу була визначена з альтернативною, послідовною та паралельною композицією, але без зв'язку. Створено модель на основі проективних послідовностей (процес задано послідовністю наближень кінцевими членами), і в цій моделі встановлено, що всі рекурсивні рівняння мають рішення. В адаптованому вигляді ця стаття була пізніше опублікована [15]. Ця алгебра процесів PA була розширена за допомогою зв'язку, щоб отримати теорію ACP (Алгебра комунікаційних процесів).

Порівнюючи три найвідоміші алгебри процесів CCS, CSP і ACP, ми можемо сказати, що в усіх трьох з них реалізовано значний обсяг роботи та додатків. Історично CCS була першою з повною теорією. На відміну від двох інших, CSP має найменш відмінну теорію рівнянь. Більше ніж два інших, ACP наголошує на алгебраїчному аспекті: існує екваціональна теорія з рядом семантичних моделей. Також ACP має більш загальну комунікаційну схему: в CCS комунікація поєднується з абстракцією, в CSP комунікація поєднується з обмеженням.

3.5 Висновки

У цьому розділі ми обговорили базові питання щодо процесу. На початку розділу ми запропонували неформальне визначення процесу та навели приклад, щоб проілюструвати використання процесу для представлення реальної системи, що аналізується.

Після цього було запропоновано класифікація процесів і формальне визначення процесу. У це визначення ми включили дві групи операторів, а саме: оператори, що керують станами процесу, і оператори, що визначають значення вихідних сигналів (подій) процесу. Таким чином, введене формальне визначення процесу об'єднує набір підалгоритмів, які повинні міститися в загальному алгоритмі процесу. Отже, для побудови оригінальної моделі системи необхідно визначити дві згадані вище групи операторів і побудувати програмну реалізацію цих операторів.

Після цього інструмент потокових графіків був наданий як один із способів представлення моделей процесу. Наприкінці цього розділу ми зробили короткий огляд історії алгебр процесів як іншого способу представлення та вивчення моделей процесів. Алгебри процесів – це узагальнений підхід до опису, представлення та вивчення поведінки паралельних або розподілених систем алгебраїчними засобами. У цих тезах ми будемо використовувати базову алгебру процесів для опису аналізованої системи

Умови виникнення ресурсних взаємоблокувань (умови Коффмана). свідчать про те, що для виникнення ресурсних взаємоблокувань повинні виконуватись чотири умови:

- умова взаємного виключення – кожен ресурс або виділений на даниq момент лише одному процесу, або доступний;

- умова утримання та очікування. – процеси, які утримують зараз раніше виділені їм ресурси, можуть вимагати нові ресурси;

- умова невивантажуваності – раніше виділені ресурси неможливо знайти примусово відібрані в процесу. Вони мають бути явно вивільнені тим процесом, який їх утримує;

- умова циклічного очікування – повинна існувати кільцева послідовність двох і більше процесів, кожен із яких очікує вивільнення ресурсу, утримуваного наступним членом послідовності.

Оскільки алгебра процесів може бути використана для опису умов Коффмана, то її можна використовувати для моделювання тупиків.

4 ОПТИМІЗАЦІЯ ПРОЦЕСНИХ МОДЕЛЕЙ СКЛАДНИХ СИСТЕМ

4.1 Вступ

Моделювання широко використовується для аналізу складних систем (CS), але для моделювання CS необхідно описати її в термінах деякої формальної схеми. Одним із можливих описів CS є процесний підхід, математична схема високого рівня абстракції.

Будь-який процес має дві складові: список чутливості та алгоритм, який виконує якусь конкретну функцію. Коли КС описується в термінах процесів, необхідно застосовувати різні реорганізації її структури, особливо на етапі оптимізації. Ця глава зосереджена на етапі оптимізації, і її основною метою є надання методу об’єднання кількох процесів в один.

Таке об’єднання процесів буде називатися «згорткою». Згортка процесів (CP) може бути використана для оптимізації моделей CIS для розподіленого моделювання на базі багатопроцесорних комп'ютерів. У цій главі ми надаємо алгоритм і формальну специфікацію роботи CP. Формальна специфікація побудована на базі базової алгебри процесів.

4.2 Визначення агрегації процесів

Агрегацію процесів можна розділити на два етапи [14]:

- складання списку чутливості (LS) нового процесу на основі списків чутливості початкових процесів;
- алгоритм об'єднання вихідних процесів в єдиний алгоритм нового процесу.

Подія, яка запускає алгоритм, вважатиметься вхідною подією, тоді як подія, що генерується під час процесу, вважатиметься вихідною. Нарешті, список чутливості буде розглядатися як набір вхідних подій.

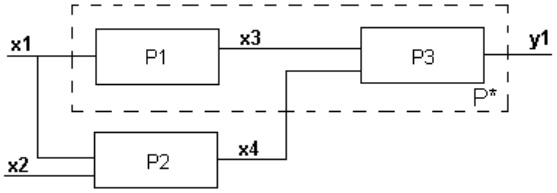

Рисунок 4.1 – Зв’язок між процесами

Зрозуміло, що список чутливості нового процесу повинен складатися з подій у списках чутливості початкових процесів. Розглянемо приклад, наведений на рисунку 5.1, в якому список чутливості процесу P1 – це x1, список чутливості процесу P2 складається з x1 і x2, а список чутливості P3 складається з x3 і x4.

Припустимо, що {1.Pi, 2.Pi, …, j.Pi} представляє список вхідних подій процесу Pi; Pi.1, Pi.2, …, Pi.j представляють вихідні події процесу Pi; а LSi – список чутливості процесу Pi, то список чутливості процесу P* можна визначити наступним чином:

$$LS^* = (LS1 \cup LS2 \cup ... \cup LSn) \quad \forall \quad i.Pm \neq Pk.j\,, \quad i.Pm \neq j.Pk$$

$$\begin{array}{l} m = \overline{1, n} \\ k = \overline{1, n}\,, \quad \text{but } k \neq m \\ i = \overline{1, \max.Pn} \\ j = \overline{1, Pk.\max} \end{array}, \qquad (4.1)$$

де відповідно до цього правила список чутливості процесу P* складається з x1, x2, x3 і x4.

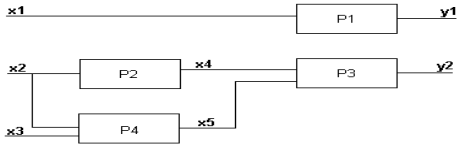

Рисунок 4.2 – Схема зв’язків між процесами

Процес в CIS може бути взаємопов’язаним (пов’язаним з іншими подіями) або ізольованим (не пов’язаним взагалі).

4.3 Агрегація ізольованих процесів

Розглянемо послідовність згортання процесів P1 і P2, зображену на рисунку 5.2. У цьому випадку процеси ізольовані, і для представлення загального алгоритму можна використовувати оператор перемикання випадків. Цей оператор визначається таким чином, що кожному випадку відповідає певна подія (або набір подій), а загальна поведінка нового процесу буде визначатися обраною подією, яка відбулася

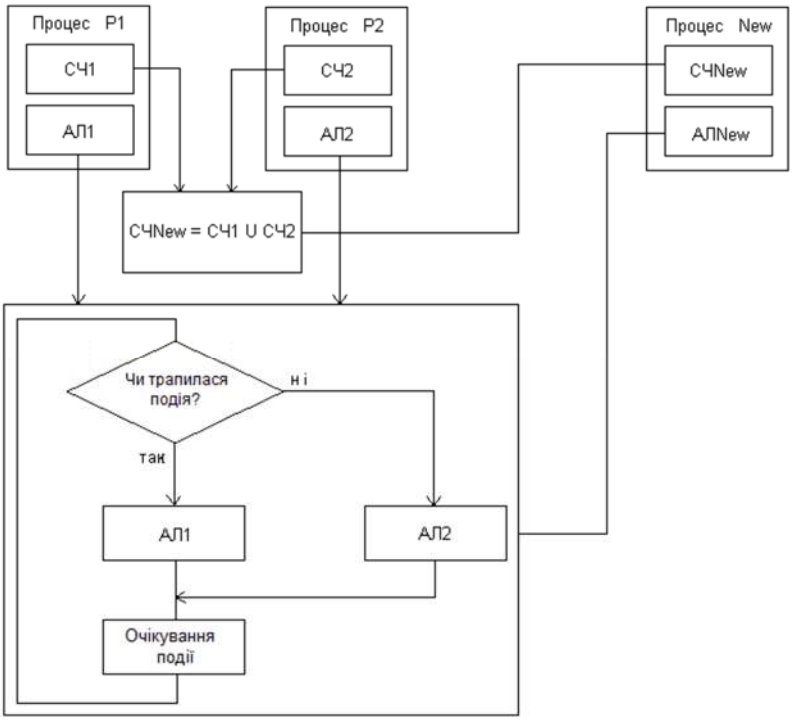

Рисунок 4.3 – Порядок об'єднання процесів P1 і P2

Отже, відповідна частина загального алгоритму буде виконуватися в залежності від обраної події. Порядок асоціації алгоритмів для P1 і P2 проілюстровано на рисунку 4.3.

4.4 Агрегація взаємопов’язаних процесів

Розглянемо послідовність згортки процесів Р2 і Р3 (див. рис. 5.2). У цьому випадку Р3 з'єднується з Р2 подією x4. Отже, Р2 і Р3 пов’язані між собою.

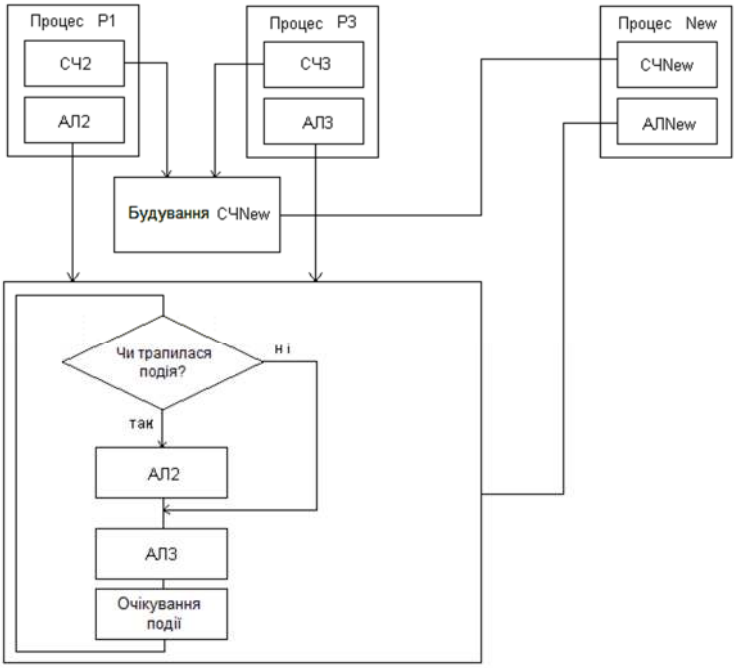

Рисунок 4.4 – Порядок агрегації процесів Р2 і Р3

Поведінка нового процесу визначатиметься подією, яка відбулася, і алгоритм буде частково або повністю виконано залежно від того, яка подія відбулася на рисунку 4.4.

4.5 Змішана агрегація

У деяких випадках може знадобитися об’єднати довільний набір процесів в один. Задіяні процеси можуть бути взаємопов’язаними, ізольованими або обома. У цьому випадку необхідно застосувати комбінацію методів ізольованої та взаємопов’язаної згортки, подібних до описаних у розділах 5.3 та 5.4 відповідно. Це випадок змішаної згортки.

Для ілюстрації цього прикладу розглянемо згортку процесів P1, P2 і P3 на рисунку 5.2. Тут загальна згортка процесів буде виконана послідовно, тобто крок за кроком, і для простоти на кожному кроці буде об’єднано лише два процеси.

Перший крок включає згортку P2 і P3, обидва пов’язані між собою подією x4 і ізольовані від P1. Отже, згортку P2 і P3 можна виконати за допомогою підходу, описаного на рисунку 5.3. процес поступливості P*, який включає список чутливості LS* і алгоритм, показаний на рисунку 5.4. Зверніть увагу, що список чутливості LS* будується згідно з правилом 5.1.

Другий крок включає згортку P* з P1, що можна зробити за допомогою підходу, описаного в 5.2., оскільки вони ізольовані.

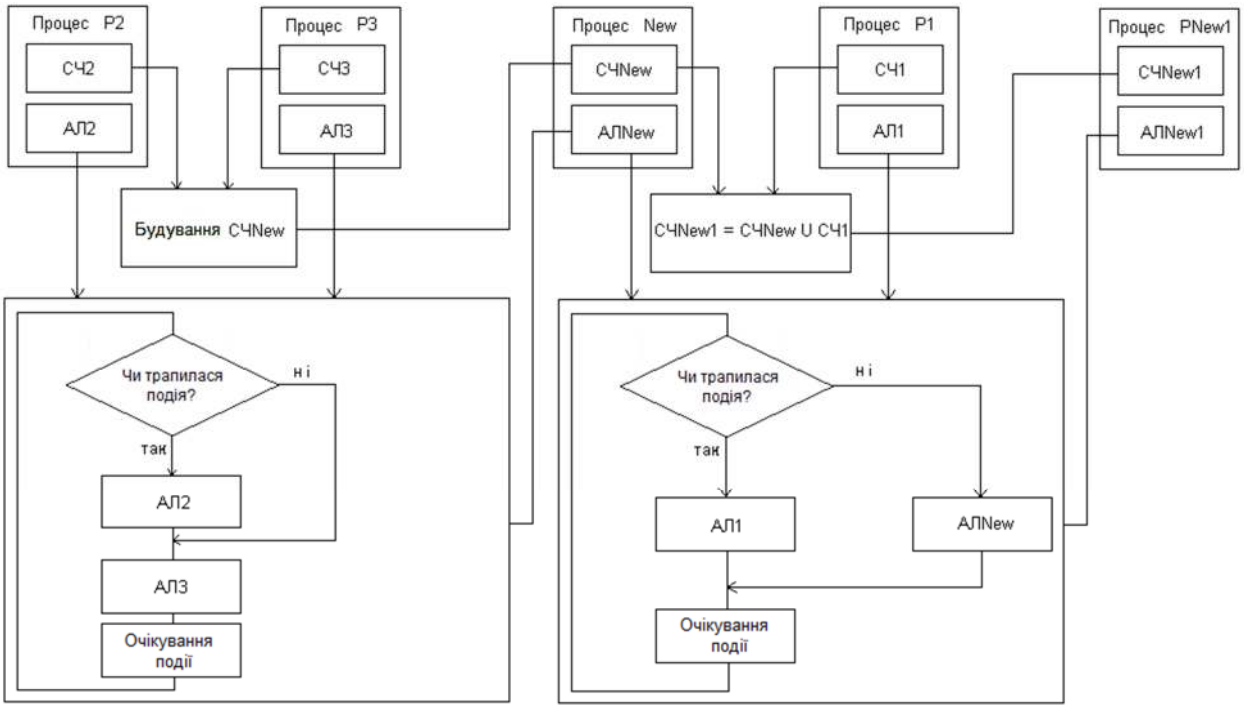

Рисунок 4.5 – Порядок згортки процесів P1, P2 і P3

Рисунок 4.5 ілюструє задіяні перетворення. Таким чином можна отримати згортку взаємопов’язаних і ізольованих процесів.

Таким чином, згортка взаємопов’язаних процесів розглядається як послідовність підалгоритмів, у яких кожен із них буде частково або повністю

виконуватися відповідно до виникнення вхідних подій, подібно до оператора if – else. З іншого боку, згортку ізольованих процесів можна розглядати як групу некорельованих підалгоритмів, які виконуються відповідно до появи вхідних подій подібним чином до оператора перемикання випадків.

Показано послідовне згортання кількох процесів в один для різних умов входу, але дії, що виконуються на етапі об'єднання алгоритмів, описані лише схематично. Однак за допомогою алгебри процесів техніку згортки процесів можна описати формально [9].

Давайте введемо оператор «вибір», який вибирає один варіант поведінки серед набору варіантів відповідно до певної вхідної умови (тобто логічного виразу). Цей оператор можна формально записати так:

$$AL1 \triangleleft condition \triangleright AL2, \tag{4.2}$$

де AL1 і AL2 – це алгоритми процесу, який потрібно згорнути, і будуть виконані, якщо умова є істинною або хибною відповідно.

Операцію вибору можна розглядати як розширення операції «+», похідної від ВРА. У цій операції, крім врахування принципу альтернативності, також вводиться алгоритм процесу. Тепер можна більш формально описати роботу асоціації.

Повертаючись до різних можливостей умов входу в асоціацію, згортання кількох процесів в один можна переформулювати таким чином.

Агрегація ізольованих процесів.

У цій ситуації характерний вибір альтернативної поведінки нового процесу.

У прикладі на рисунку 4.2 поведінку нового процесу можна формалізувати таким чином:

$$AL1 \triangleleft x1 \triangleright AL2 \tag{4.3}$$

Загалом, замість x1 може бути атрибут появи події(ів) зі списку чутливості початкового процесу P1.

Агрегація процесів із спільними подіями

Це той випадок, коли процеси, які потрібно згорнути, мають спільні події у своєму списку чутливості. Як правило, алгоритми цих процесів повинні виконуватися паралельно. Формалізацію такої поведінки можна здійснити за допомогою операції ||.

На прикладі згортки процесів P2 і P4, зображеному на рисунку 5.2, цю операцію можна записати так:

$$AL2 \,||\, AL4 \triangleleft x2 \triangleright NULL \tag{4.4}$$

Загалом, x2 може представляти атрибут появи загальної(их) події(ій) у списках чутливості початкових процесів. У правій частині рівняння (4.4) стоїть NULL, але загалом може бути більше операторів вибору. Розглянемо приклад на рисунку6, на якому P1 і P2 мають незалежні події x1 і x2, відповідно, на додаток до спільної події x2.

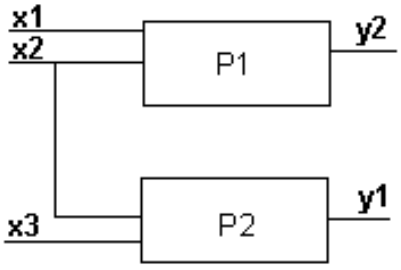

Рисунок 4.6 – Схема зв’язків між процесами

Таким чином, алгоритми P1 і P2 повинні працювати паралельно лише у випадку настання події x2. В іншому випадку вони повинні працювати відповідно до моментів настання подій x1 і x3. Таку логіку роботи можна формалізувати так:

$$AL1 \,||\, AL2 \lhd x2 \rhd [AL1 \lhd x1 \rhd AL2] \tag{4.5}$$

Агрегація взаємопов’язаних процесів

Типовим у цій ситуації є те, що повинен виконуватися лише один процес або їх послідовність залежно від вхідної події.

Розглядаючи згортку процесів P2 і P3 на рисунку 4.2, формалізацію асоціації алгоритмів можна виразити у вигляді:

$$AL2 * AL3 \lhd x2 \rhd AL3 \tag{4.6}$$

Оператор * дозволяє нам описувати послідовні процеси, тобто коли один процес повинен виконуватися після іншого.

Загальна агрегація процесів

Це ситуація, коли загальний процес включає багато різних комбінацій прикладів, згаданих вище.

Для ілюстрації цього випадку розглянемо згортку процесів P1, P2 і P3 (рисунок 5.2). На першому етапі алгоритми P2 і P3 згортаються таким чином:

$$AL2 * AL3 \lhd x2 \rhd AL3 \tag{4.7}$$

Результат (4.7) стає новим процесом P*. Потім на другому етапі алгоритм P* і P1 агрегуються з:

$$AL1 \lhd x1 \rhd [AL2 * AL3 \lhd x2 \rhd AL3] \tag{4.8}$$

Таким чином можна побудувати систематичний механізм асоціації, що включає всі можливості, що з’являються з будь-якою довільною кількістю комбінацій. Це узагальнення представлено на рисунку 4.7 у вигляді потокового графа, де використано наступне позначення:

- P1: Перший процес;
- P2: другий процес;
- LS1: список чутливості P1;
- LS2: список чутливості P2;
- M: набір спільних елементів-подій для LS1 і LS2;
- S1: набір елементів-подій, створених P1, які входять до P2;
- S2: набір елементів-подій, створених P2, які входять до P1;
- R1: набір елементів-подій, які є тільки в списку чутливості P1;
- R2: набір елементів-подій, які є тільки в списку чутливості P2;
- P [S]: атрибут появи події (або подій) із множини S.

4.6 Опис алгоритму агрегації

Розглянемо покроково порядок агрегації процесів за алгоритмом.

Крок 1. Початок агрегації. Підготовка вхідної інформації для злиття алгоритмів вихідних процесів.

Крок 2. Перевірте, чи всі процеси вже об’єднані чи ні. Якщо вони є, згортка завершується, а отримані результати передаються програмі для подальшої обробки, інакше вибирається інша пара процесів для об’єднання.

Крок 3. Перевіряються списки чутливості, і, якщо список чутливості хоча б для одного процесу порожній, видається відповідне попередження.

Крок 4. Побудовано множини M, S1 і S2. Вони використовуються для визначення конфігурації початкових процесів у новому процесі.

Крок 5. Після цього необхідно перевірити множину M. Цей набір включає події, присутні в списках чутливості обох процесів.

Наявність таких подій у множині M означає, що при їх появі в процесі моделювання алгоритми першого і другого процесів повинні виконуватися одночасно.

Використовуючи результат перевірки вмісту множини M, алгоритм відносить процеси, що зливаються, або до групи, де одночасне виконання неможливе, або до групи, де така ситуація можлива.

Крок 6. Розглянемо випадок, коли M – порожня множина, тобто списки чутливості процесів, що об’єднуються, не містять ідентичних подій і одночасне виконання алгоритмів неможливе.

Крок 7. Розраховано вміст множин R1 і R2. Ці набори використовуються для визначення способу об'єднання алгоритмів для вихідних процесів.

R1 містить події, які впливають тільки на роботу першого процесу і визначаються відповідно до наступного виразу:

$$R1 = LS1 – S2, \tag{4.9}$$

де $S2 = P2OUT \cap LS1$; P2OUT – множина елементів-подій, згенерованих P2; R2 містить події, які впливають лише на другий процес і визначаються за наступним виразом

$$R2 = LS2 – S1, \tag{4.10}$$

де $S1 = P1OUT \cap LS2$.

Крок 8. За допомогою наборів S1, S2, R1 і R2 визначається правило об'єднання алгоритмів вихідних процесів. Слід згадати випадок, коли множини S1 і S2 непорожні, а множини R1 і R2 порожні.

У цьому випадку виникає помилка, оскільки списки чутливості вихідних процесів не містять подій, які передаються їм ззовні або від інших процесів. Така ситуація неможлива і розглядається схемою як помилка.

Крок 9. Розглянемо випадок, коли M не є порожньою множиною, тобто списки чутливості процесів, що зливаються, містять ідентичні події.

Крок 10. Аналогічно виконується обчислення вмісту наборів R1 і R2, які використовуються для визначення способу злиття алгоритмів вихідних

процесів. R1 містить події, які впливають лише на роботу першого процесу, але цей набір визначається за іншим виразом:

$$R1 = LS1 – S2 – M, \quad (4.11)$$

де $S2 = P2OUT \cap LS1$; $M = LS1 \cap LS2$.

R2 містить події, які впливають тільки на роботу другого процесу. Ця множина визначається відповідно до наступного виразу:

$$R2 = LS2 – S1 – M, \quad (4.12)$$

де $S1 = P1OUT \cap LS2$; $M = LS1 \cap LS2$.

Крок 11. Використовуючи вміст множин S1, S2, R1 і R2, визначено правило об’єднання алгоритмів вихідних процесів.

Слід зазначити, що правила побудови алгоритмів можуть бути однаковими для різних випадків.

Наприклад, якщо $S2 = 0$, $R1 = 0$, $R2 = 0$ і якщо $S2 \neq 0$, $R1 = 0$, $R2 = 0$ те саме правило ALP1 || Застосовується ALP2.

4.7 Висновки

Формальна операція асоціації інформаційних процесів була визначена під назвою «агрегація» в рамках базової алгебри процесів і оператора «вибір». Результатом є блок-схема, що показує алгоритм згортки процесів для будь-яких довільно можливих умов входу. При такому підході можна здійснити програмну реалізацію даного оператора для підсистеми структурної оптимізації моделей процесів у SAD.

Згортка процесів може бути використана для оптимізації моделей вихідних процесів КІС з метою досягнення ефективного розподіленого моделювання на базі багатопроцесорного комп’ютера. По суті, для N процесорів мета полягає в тому, щоб реорганізувати вихідне представлення Комплексної системи таким чином, щоб максимізувати обчислювальну ефективність, що можна зробити за умови, що кожен процесор призначений для ізольованого набору процесів, які могли виникнути в результаті операції агрегації.

5 ОЦІНКА ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНОГО ПІДХОДУ

5.1 Цілі і план оцінки ефективності запропонованого підходу

Виконати аналіз процесної моделі, щоб виявити зв’язки між компонентами моделі.

Оцінити ефективність використання запропонованих підходів за допомогою програмних засібів.

Для побудови таких моделей та отримання результатів ми використовували систему моделювання GPSS World. Це досить простий і зручний інструмент для ефективної розробки моделей для широкого циклу реальних систем.

5.2 Планування експерименту

У рамках експерименту ми хочемо отримати час моделювання для аналізованої системи до та після реорганізації. Необхідно виконати наступне завдання:

Змоделювати аналізовану систему до та після реорганізації відповідно до запропонованих підходів.

Вихідні дані для експерименту такі:

- кількість процесорів: 1, 2, 4, 8, 16;
- час виконання процесу: 100 (у модельний час);
- час перемикання між процесами: 30 (модельний час);
- кількість ступенів: 100.

У результаті ми повинні отримати час моделювання для кожного випадку та побудувати графіки, де всі ці результати мають бути проілюстровані.

5.3 Структура аналізованих систем, GPSS моделі та результати моделювання

Перша система така, що всі процеси в цій системі паралельні. Модель GPSS для цього випадку за вихідними даними (кількість процесорів 16) можна представити так (перший – до реорганізації, другий – після реорганізації):

Лістинг 5.1 – Оформлення моделі GPSS

```
PRNUM      VARIABLE 1000
TPER       VARIABLE 100
TSWITCH    VARIABLE 30
TRES       VARIABLE V$TPER+V$TSWITCH
PPP        STORAGE  16
GENERATE     1,,,V$PRNUM
QUEUE        OOO
ENTER        PPP
DEPART       OOO
ADVANCE      V$TRES
LEAVE        PPP
TERMINATE    1
START 1000

EXTIMES    VARIABLE 10
PRNUM      VARIABLE 16#V$EXTIMES
TPER       VARIABLE 600
TSWITCH    VARIABLE 30
TRES       VARIABLE V$TPER
PPP        STORAGE  16
GENERATE     1,,,V$PRNUM
QUEUE        OOO
ENTER        PPP
DEPART       OOO
ADVANCE      V$TRES
LEAVE        PPP
TERMINATE    1
START 160
```

Ми використовуємо в моделі спеціальні змінні, і за допомогою цих змінних модель може бути адаптована для випадку з заданою кількістю процесорів, процесів і так далі. Отже, результати, отримані під час моделювання, проілюстровані на рисунках 5.1 і 5.2.

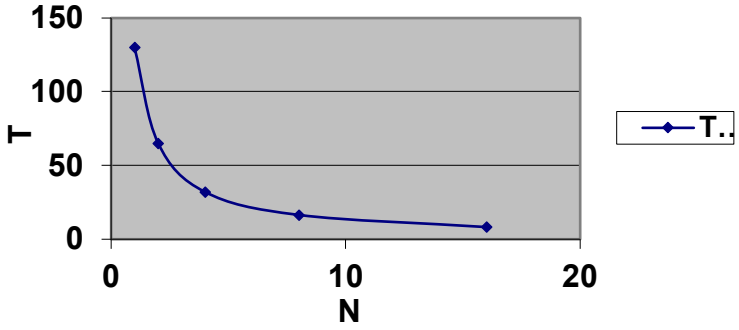

Рисунок 5.1 – Залежність часу моделювання від кількості процесорів для паралельної системи до реорганізації

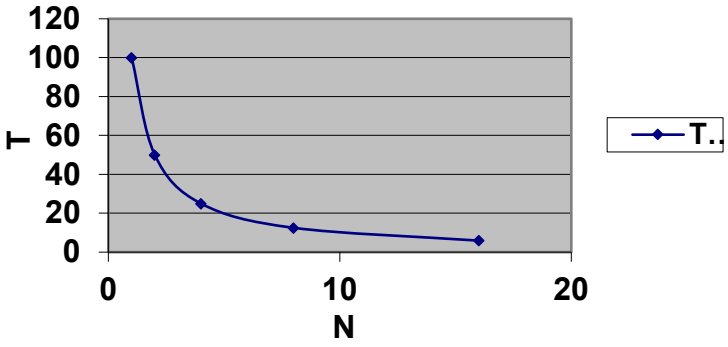

Рисунок 5.2 – Залежність часу моделювання від кількості процесів для паралельної системи після реорганізації

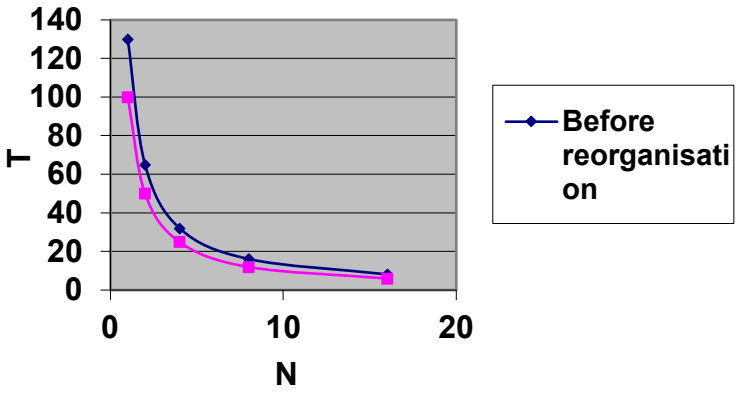

Рисунок 5.3 – Залежність часу моделювання від кількості процесів для обох паралельних систем

Друга система така, що всі процеси в цій системі є послідовними. Модель GPSS для цього випадку за вихідними даними можна представити так (перша – до реорганізації, друга – після реорганізації):

Лістинг 5.2 – Оформлення моделі GPSS

```
EXTIMES    VARIABLE 10
PRNUM      VARIABLE 16#V$EXTIMES
TPER       VARIABLE 625
TSWITCH    VARIABLE 30
TRES       VARIABLE V$TPER+V$TSWITCH
PPP        STORAGE  1
GENERATE     1,,,V$PRNUM
QUEUE        OOO
ENTER        PPP
DEPART       OOO
ADVANCE      V$TRES
LEAVE        PPP
TERMINATE    1
START 160


EXTIMES    VARIABLE 10
PRNUM      VARIABLE 16#V$EXTIMES
TPER       VARIABLE 625
TSWITCH    VARIABLE 30
TRES       VARIABLE V$TPER
PPP        STORAGE  1
GENERATE     1,,,V$PRNUM
QUEUE        OOO
ENTER        PPP
DEPART       OOO
ADVANCE      V$TRES
LEAVE        PPP
TERMINATE    1
START 160
```

Результати, отримані під час моделювання, проілюстровані на рисунку 5.3 і 5.4.

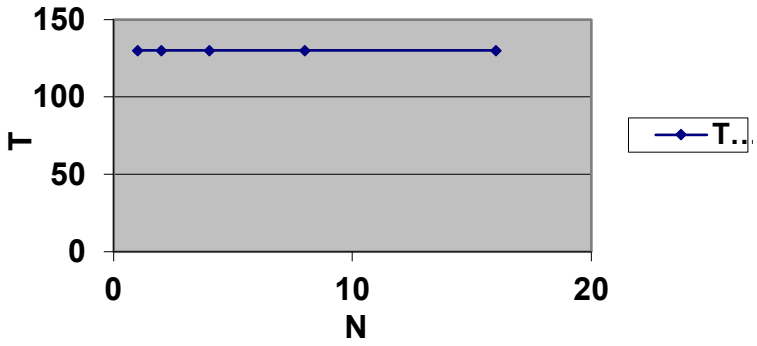

Рисунок 5.4 – Залежність часу моделювання від кількості процесів для послідовної системи до реорганізації

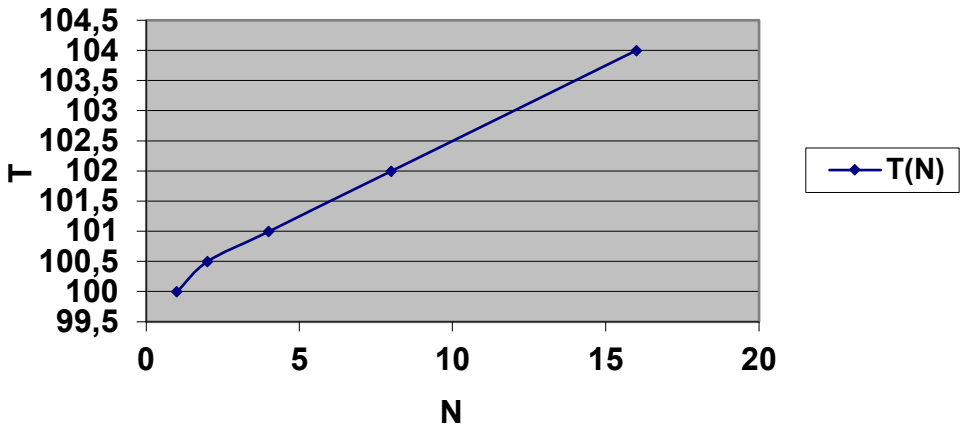

Рисунок 5.5 – Залежність часу моделювання від кількості процесів для послідовної системи після реорганізації.

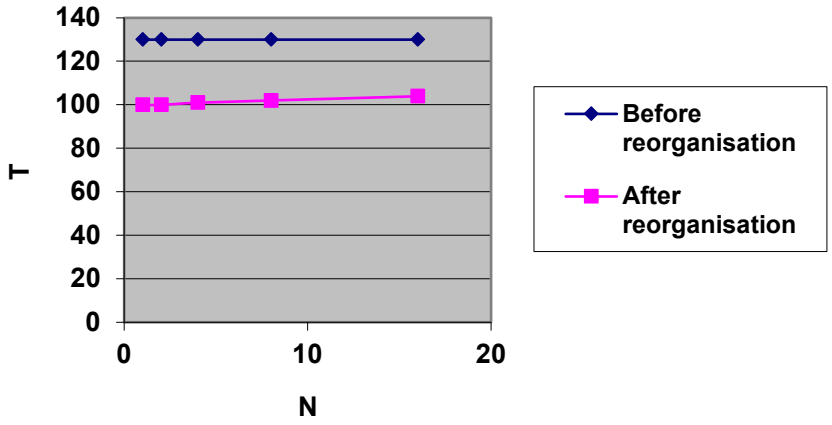

Рисунок 5.6 – Залежність часу моделювання від кількості процесів для обох послідовних систем

5.4 Аналіз результатів

Для оцінки отриманих результатів необхідно подивитися на рис. 6.4. На цьому рисункуми можемо побачити графік залежності часу моделювання від кількості процесорів для обох систем: до і після моделювання.

Порівнюючи малюнки 5.1 і 5.2 ми бачимо, що час моделювання залежить від кількості процесів системі після реорганізації менший, ніж час моделювання системи до реорганізації. Це можна пояснити тим, що кількість перемикань між процесами досить інтенсивна. Отже, час моделювання збільшується разом із інтенсивним перемиканням між процесами. Така ж ситуація з системою, в якій усі процеси послідовні. Але в цьому випадку час моделювання для системи до реорганізації є постійним, оскільки кожен процес очікує завершення процесу раніше. У цьому випадку ми можемо спостерігати дивну ситуацію: час моделювання збільшується разом із кількістю процесорів.

Таким чином, результати, отримані під час моделювання, доводять, що наш підхід до виявлення паралелізму в структурі моделі разом із операцією «згортка» повинен дати нам вражаючі результати в практичному використанні.

ВИСНОВКИ

В роботі було зосереджено на аналізі інформаційних процесів, представлених у формі математичних схем. Робота досліджувала формальні інструменти для представлення моделей процесів. В результаті було знайдено два методи відповідного представлення моделей процесів: потокові графи для процесів і основні операції алгебри процесів. При необхідності можна переходити від одного представлення до іншого. У другому розділі ми представили математичну схему, що описує інформаційні процеси. Розроблено математичну модель процесів функціонування на основі математичної агрегаційної моделі, яка є ще однією схемою високого рівня абстракції. Досліджено проблему розподіленого моделювання складних інформаційних систем та запропоновано етапи підготовки вихідних моделей для розподіленого моделювання. Крім того, запропоновано алгоритм для виявлення внутрішнього паралелізму в структурі моделей процесів. Після цього була досліджена проблема адаптації вихідної моделі до певної апаратної платформи. В результаті ми розробили ще один алгоритм для об'єднання деяких процесів в універсальний процес. Ця операція злиття була названа операцією згортки.

Після створення теоретичної частини дисертації було реалізовано комп’ютерне програмне забезпечення із запропонованими методами. Для розробки такої програми ми використовували мову C++ і бібліотеку STL. Ці інструменти дозволяють прискорити розробку програми. Для оцінки адекватності отриманих результатів ми протестували та проаналізували роботу програми. За результатами тестування програма працює коректно.

Таким чином, в рамках даної кваліфікаційної роботи було розроблено два алгоритми та запропоновано програму, яка їх реалізує.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Таненбаум, Е. Сучасні операційні системи [Текст]: навч. посібник / Таненбаум Е. – 1009 с.

2. Дейт, К.Дж. Введення до систем баз даних [Текст]: навч. посібник / Дейт К.Дж. – М. : Вильямс, 2000. – 378 с.

3. Bodyagin, I. Deadlocks [Електронний ресурс]. – Режим доступу : http://www.rsdn.article/.

4. Milner, R, The Space and Motion of Communicating Agents [Електронний ресурс]. – Режим доступа : http://www.ocr.com.

5. Using critical section objects [Электронный ресурс]. – Режим доступу : www.msdn.microsoft.com.

6. Holt, R. Some Deadlock Properties of Computer Systems [Text] / R. Holt // ACM Computing Surveys. – 1972. – № 3. – pp. 179-196.

7. Дейкстра, Е. Взаємодія послідовних процесів [Текст] : сборник статей / Є. Дейкстра // Мови програмування. – М. : Мир, 1972. – с. 56-65.

8. Coffman, E., Elphick, M., and Shoshani, A. System Deadlocks [Text] / E. Coffman, M. Elphick, A. Shoshani // ACM Computing Surveys. – 1971. – № 3(2). –pp.67-78.

9. Even, S. Graph Algorithms, Potomac, MD [Text] / S. Even // Computer Science Press. – 1979. - № 4. – pp. 114-121.

10. Leibfried Jr., T. F. A deadlock Detection and Recovery Algorithm Using the Formalism of a Directed Graph Matrix [Text] / T. F. Leibfried Jr. // ACM SIGOPS Operating Systems Review. – 1989. – № 23(2). – pp. 45-55.

11. Gonzaa, J. R. A Distributed Deadlock Resolution Algorithm for the AND Model [Text] / J. R. Gonzaa, F. Farina, J. R. Garitagoitia, C. F. Alastruey, J. M. Bernabeu-Auban // IEEE Trans. on Parallel and Distributed Systems. – 1999. – №10. – pp 433-447.

12. Knapp, E. Deadlock Detection in a Distributed databases [Text] / E.

Knapp // ACM Computing Surveys. – 1987. – № 9. – pp. 303-328.

13. Singhal, M. Deadlock Detection in Distributed Systems [Text] / M. Singhal // Computer. – 1989. – №4. – pp. 37-48.

14. Горбачов В.О., Мантуров Д. О. Метод виявлення взаємного блокування в інформаційних системах. Одинадцята міжнародна науково-технічна конференція ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ, ХНУРЕ, листопад 2023 р.

15. Kshemkalyani, A.D. Invariant-Based Verification of a Distributed Deadlock Detection Algorithm [Text] / A.D. Kshemkalyani, M. Singhal // IEEE Trans. Software Eng. – 1991. – № 8(17). – pp. 789-799.

16. GonzaÂlez de MendõÂvil, J.R. Correctness of a Distributed Deadlock Resolution Algorithm for the Single Request Model [Текст] / J.R. GonzaÂlez de MendõÂvil, J. Bernabeu, A. Demaille, and J.R. Garitagoitia // Proc. Third Euromicro Workshop Parallel and Distributed Systems – 1995 – № 13 – pp. 254-261.

17. Kshemkalyani A.D. On Characterization and Correctness of Distributed Deadlock Detection [Текст] / A.D. Kshemkalyani and M. Singhal // Technical Report OSU-CISRC-6/90 TR – 1991 – № 15 – pp. 123-145.

18. Kshemkalyani A.D. Characterization and Correctness of Distributed Deadlock Detection and Resolution [Текст] : dissertation PhD : 08.00.13 : защищена 12.02.02 : утв. 24.06.02 / A.D. Kshemkalyani - Ohio State Univ, 1991, pp. 234