Хмельницький національний університет
Факультет програмування та комп'ютерних і телекомунікаційних систем
Кафедра кібербезпеки та комп'ютерних систем і мереж

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Інформаційна технологія діагностування КС на наявність ваємоблокувань процесів
Назва теми

Галузь знань 12 – Інформаційні технології

Спеціальність 123 – Комп'ютерна інженерія

КРМКІ. 170160.19.01.16 ПЗ

Виконав: студент 2 курсу, група КІ1м-19-1 Горчилов І.І.
Підпис

Керівник проф., д. т. н, професор кафедри КбКСМ Андрощук О.С.
Підпис

Нормоконтролер доц., к. т. н, доцент кафедри КбКСМ Муляр І.В.
Підпис

До захисту допускаю:
Зав. кафедри КбКСМ, к.т.н., доцент Кльоц Ю.П.
Підпис

10.12 2020 р.

Хмельницький, 2020

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем

Кафедра Кафедра кібербезпеки та комп'ютерних систем і мереж

Освітній рівень магістр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма Програмування та захист комп`ютерних систем і мереж

ЗАТВЕРДЖУЮ

Зав. кафедри Ю.П.Кльоц

" 01" 09 2020 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Горчилову Івану Івановичу

Прізвище, ім'я, по батькові студента

1. Тема роботи Інформаційна технологія діагностування КС на наявність ваємоблокувань процесів

Керівник роботи Андрощук О.С,

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

доктор технічних наук, професор

Затверджена наказом № 118 ректора університету додаток №23 від 01.09.2020

2. Строк подання студентом проекту (роботи) на кафедру 20.11.2020

3. Вихідні дані до проекту (роботи) взаємоблокування, методи вирішення задачі, прогнозування, комп'ютерні системи

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Вступ. Дослідження виконання процесів у комп'ютерних систем та методів вирішення взаємоблокувань. Математична модель прогнозування взаємоблокувань. Метод прогнозування взаємоблокувань процесів з метою підвищення надійності функціонування комп'ютерних систем. Апробація ефективності запропонованого методу і алгоритмів його реалізації. Висновки.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Загальна характеристика магістерської роботи. Дослідження та методів вирішення взаємоблокувань. Математична модель методу. Основні положення методу. Алгоритмічна реалізація методу. Апробація методу – вхідні дані. Апробація методу - результати моделювання. Висновки.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Муляр І.В. доцент кафедри КбКСМ		

7. Дата видачі завдання «_2_» _____вересня_____ 2020р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Вибір напряму дослідження та узгодження тематики КРМ з керівником	2.02.2020	
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	2.03.2020	
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	1.04.2020	
4	Робота над розділом 2 – розробка моделей і методів для вирішення поставленої задачі	1.05.2020	
5	Робота над науковою публікацією	1.06.2020	
6	Уточнення і затвердження теми	1.09.2020	
7	Робота над розділом 3 – розробка методу та алгоритмів, їх аналіз	2.09.2020	
8	Робота над розділом 4 – апробація запропонованих рішень	1.10.2020	
9	Узгодження отриманих результатів; оформлення пояснювальної записки згідно вимог	1.11.2020	
10	Оформлення графічної частини	8.11.2020	
11	Попередній захист роботи	10.11.2020	
12	Захист роботи на засіданні ЕК	5.12.2020	

Студент _____ _____
Підпис Ініціали, прізвище

Керівник роботи _____ _____
Підпис Ініціали, прізвище

4

АНОТАЦІЯ

Тема кваліфікаційної роботи: Інформаційна технологія діагностування КС на наявність взаємоблокувань процесів

Автор роботи: Горчилов Іван Іванович

Керівник роботи: д.т.н., проф. Андрощук О.С.

Загальний обсяг роботи: 141 сторінка, 31 рисунок, 6 таблиць, 2 додатки, 109 посилань.

ВЗАЄМОБЛОКУВАННЯ, СТАНИ ПРОЦЕСІВ, ПРОГНОЗУВАННЯ ВЗАЄМОБЛОКУВАНЬ, КОМП'ЮТЕРНІ СИСТЕМИ

Метою кваліфікаційної роботи є розроблення методу, алгоритмів та відповідних їм програмних засобів прогнозування стану процесів, що виконуються в комп'ютерних системах для підвищення надійності роботи КС.

Дана кваліфікаційна робота присвячена розробці методу прогнозування взаємоблокувань процесів із використанням сигнатур процесів та граничного стану для підвищення надійності роботи комп'ютерних систем. Розроблений метод дозволяє визначити множину процесів, що наближаються до взаємоблокування, та скоротити часові витрати на виконання процесів в 2,3 рази в порівнянні із стандартними методами.

Дата

Підпис студента

5

ANNOTATION

a qualification work of Horchylov Ivan

entitled «A method of creating a shift cipher using optimal Huffman coding to increase the efficiency of protecting computer systems.».

Mentor: Ph.D. Androschyk O.

Total volume of work: 141 pages, 31 figures, 6 tables, 2 appendices, 109 references.

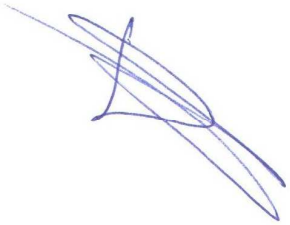
DEADLOCKS, PROCESS STATUS, PREDICTION OF DEADLOCKS, COMPUTER SYSTEMS

The purpose of the qualification work is to develop a method, algorithms and related software for forecasting the status of processes performed in computer systems to improve the reliability of the COP.

This qualification work is devoted to the development of a method for predicting the interlocking of processes using process signatures and limit state to improve the reliability of computer systems. The developed method allows to determine the set of processes approaching mutual blocking, and to reduce time expenses for performance of processes in 2,3 times in comparison with standard methods.

Date

Signature



ЗМІСТ

ВСТУП

Актуальність дослідження. В процесі експлуатації персональних комп'ютерів (ПК) досить часто виникає ситуація блокування процесів, що виконуються на них. Причинами такого явища є помилки у програмному коді ПЗ, невідповідність ПЗ його специфікації. Наслідком такої ситуації є вихід з ладу апаратних складових, некоректне звертання програмного забезпечення (ПЗ) до пам'яті, конфліктні ситуації між різним ПЗ.

Для визначення і усунення причин і наслідків блокування процесів у комп'ютерній системі використовують різноманітні методи і засоби, а саме: семафори, монітори, м'ютекси, спеціалізовані діагностичні програми для перевірки програмної та апаратної цілісності ПК, тестери для перевірки справності апаратної частини. Проте на практиці їх не завжди доцільно застосовувати для вирішення задачі блокування.

Однією із проблем виконання програм на ПК є проблема уникнення входження в стан взаємоблокування. Дані питання досліджувались Дейкстрою, Хоаром, Брінч-Хансеном, Деккером, Петерсеном, Коффманом, Хольтом. Розроблено багато методів та алгоритмів для уникнення взаємоблокувань процесів. Проте частина з них не може бути реалізованою в сучасних операційних системах і носить теоретичний характер. Інша частина при реалізації стає досить громіздкою в структурі операційної системи. Тому розробники ОС Windows та Unix не включали відомі алгоритми уникнення взаємоблокувань процесів. За умови виконання в системі не великої кількості процесів відсутність таких засобів була допустима. Стрімкий розвиток апаратних засобів ПК, зростання об’єму та складності ПЗ та розв’язуванням на ПК масштабніших відповідальніших задач не повинно дозволяти стану взаємоблокування, що в свою чергу вимагає розробки нових підходів до вирішення задачі блокування процесів.

Мета кваліфікаційної роботи полягає в розробленні методу, алгоритмів та відповідних їм програмних засобів прогнозування стану процесів, що виконуються в комп’ютерних системах, для підвищення надійності роботи КС.

Для досягнення мети роботи необхідно вирішити наступні завдання:

1) провести аналіз процесів та їх взаємодії у сучасних операційних системах як об’єктів діагностування та виявити проблеми, що виникають у процесі їх роботи;

2) провести аналіз відомих методів вирішення задачі взаємоблокування процесів з метою дослідження та систематизації їх недоліків;

3) розробити метод прогнозування стану процесів в ПК, який на відміну від відомих дозволив би уникати взаємоблокування процесів та був простим у реалізації у сучасних ОС;

4) розробити методику та алгоритми визначення процесів, що наближаються до стану взаємоблокування, на основі прогнозування їх стану;

5) реалізувати метод прогнозування стану процесу у вигляді програмних додатків та впровадити їх з метою підвищення безвідмовної роботи процесів, що виконуються на персональних комп’ютерах.

Об’єктом дослідження є процес прогнозування стану взаємоблокування процесів в комп'ютерних системах

Предметом дослідження є методи та засоби прогнозування стану взаємоблокування процесів в комп’ютерних системах.

Методи дослідження. Для розв’язання поставлених задач використовуються основні положення методів аналізу даних, нечіткої логіки, теорії графів, теорії множин.

Наукова новизна одержаних результатів.

У результаті дослідження розв'язано актуальну науково-практичну задачу розроблення методу прогнозування стану взаємоблокування

процесів в комп'ютерних системах. При цьому одержано такі наукові результати:

1) Проведено аналіз процесів та їх взаємодії у сучасних операційних системах. Виявлено, що при взаємодії процесів виникає проблема взаємного блокування їх роботи.

2) Проведено аналіз відомих методів вирішення задачі взаємоблокування процесів та виявлено їхні недоліки (блокування роботи ОС, наявність циклів активного очікування, складність програмної реалізації для багатьох процесів, необхідність використання спеціалізованої команди процесора).

3) Розроблено сигнатурну модель процесу, як об’єкту дослідження.

4) На основі сигнатурної моделі процесу розроблено метод та алгоритми прогнозування стану процесу в персональному комп’ютері.

5) На основі методу та алгоритмів прогнозування стану процесу реалізовано програмне забезпечення для сучасних операційних систем для підвищення безвідмовної роботи процесів, що виконуються на персональних комп’ютерах.

Апробація роботи. Наукові результати і основні положення кваліфікаційної роботи магістра доповідались і обговорювались на всеукраїнській і міжнародній науково-практичних конференціях.

Публікації. За темою кваліфікаційної роботи опубліковано 1 стаття у збірнику наукових праць.

1 АНАЛІЗ ВІДОМИХ МЕТОДІВ УНИКНЕННЯ ВЗАЄМОБЛОКУВАНЬ ПРОЦЕСІВ

1.1 Багатозадачність та взаємоблокування процесів

На сьогодні комп'ютерні системи (КС) використовуються у різних сферах діяльності. Це підвищує вимоги до надійності КС. Оскільки однією із складових комп'ютерних систем є операційні системи (ОС), то надійність їх роботи впливає на надійність КС. Тому однією із задач ОС є забезпечення надійної роботи процесів. Проте при виконанні задач можуть виникати такі ситуації, при яких два і більше процеси весь час знаходяться в стані взаємоблокування, що негативно впливає на надійність КС. Також до складу КС входить і апаратна платформа, особливості реалізації якої також впливають на надійність КС [1] та на особливості реалізації відповідної ОС [2-9].

1.1.1 Огляд та класифікація типів ОС

Операційні системи можуть відрізнятися особливостями реалізації внутрішніх алгоритмів керування основними ресурсами комп'ютера (процесорами, пам'яттю, пристроями), особливостями використаних методів проектування, типами апаратних платформ, областями використання й багатьма іншими властивостями.

Від ефективності алгоритмів керування ресурсами комп'ютера багато в чому залежить ефективність всієї ОС у цілому. Залежно від особливостей використаного алгоритму керування процесором, операційні системи поділяють на багатозадачні й однозадачні, багатокористувацькі й однокористувацькі, на системи, що підтримують багатониткову обробку й не підтримують її, на багатопроцесорні й однопроцесорні системи [2-6, 10-13].

За кількістю одночасно виконуваних завдань операційні системи можуть бути розділені на два класи: однозадачні і багатозадачні. Однозадачні ОС в основному виконують функцію надання користувачеві віртуальної машини, роблячи більш простим і зручним процес взаємодії користувача з комп'ютером. Однозадачні ОС включають засоби керування периферійними пристроями, засоби керування файлами, засоби спілкування з користувачем. Багатозадачні ОС, крім перерахованих вище функцій, керують розподілом спільно використовуваних ресурсів, таких як процесор, оперативна пам'ять, файли й зовнішні пристрої. Очевидно, що в однозадачних системах взаємоблокування процесів неможливе, тому в подальшому будемо розглядати лише багатозадачні ОС.

За кількістю одночасно працюючих користувачів ОС поділяються на однокористувацькі та багатокористувацькі. Головною відмінністю багатокористувацьких систем від однокористувацьких є наявність засобів захисту інформації кожного користувача від несанкціонованого доступу інших користувачів. Варто відмітити, що не кожна багатозадачна система є багатокористувацькою, і не кожна однокористувацька ОС є однозадачною. Отже, кількість одночасно працюючих користувачів в багатозадачних системах не впливає на взаємоблокування процесів.

Найважливішим поділюваним ресурсом є процесорний час. Спосіб розподілу процесорного часу між декількома одночасно існуючими в системі процесами (або нитками) багато в чому визначає специфіку ОС. Серед безлічі існуючих варіантів реалізації багатозадачності можна виділити дві групи алгоритмів: невитісняюча багатозадачність та витісняюча багатозадачність. Основним розходженням між витісняючими й невитісняючими варіантами багатозадачності є ступінь централізації механізму планування процесів. У першому випадку механізм планування процесів цілком зосереджений в операційній системі, а в другому - розподілений між системою й прикладними програмами.

Ще однією важливою властивістю ОС є відсутність або наявність у ній засобів підтримки багатопроцесорної обробки - мультипроцесування. Мультипроцесування приводить до ускладнення всіх алгоритмів керування ресурсами. Багатопроцесорні ОС можуть класифікуватися за способом організації обчислювального процесу в системі із багатопроцесорною архітектурою: асиметричні ОС і симетричні ОС. Асиметрична ОС цілком виконується тільки на одному із процесорів системи, розподіляючи прикладні завдання по інших процесорах. Симетрична ОС повністю децентралізована й використовує всю множину процесорів, розділяючи їх між системними й прикладними завданнями.

Важливий вплив на вигляд операційної системи в цілому, на можливості її використання в тій або іншій області роблять особливості й інших підсистем керування локальними ресурсами - підсистеми керування пам'яттю, файлами, пристроями вводу-виводу.

На властивості операційної системи безпосередній вплив роблять апаратні засоби, на які вона орієнтована. За типом апаратури розрізняють операційні системи персональних комп'ютерів (ПК), міні-комп'ютерів, мейнфреймів, кластерів і мереж ЕОМ. Серед перерахованих типів комп'ютерів можуть зустрічатися як однопроцесорні варіанти, так і багатопроцесорні. У кожному разі специфіка апаратних засобів, як правило, відбивається на специфіці операційних систем.

Очевидно, що ОС мейнфреймів є більш складною й функціональною, ніж ОС персонального комп'ютера. Так, в ОС мейнфреймів функції по плануванню потоку виконуваних завдань реалізуються шляхом використання складних пріоритетних дисциплін і вимагають більшої обчислювальної потужності, ніж в ОС персональних комп'ютерів.

Багатопроцесорні системи вимагають від операційної системи особливої організації, за допомогою якої сама операційна система, а також підтримувані нею додатки могли б виконуватися паралельно окремими

процесорами системи. Паралельна робота окремих частин ОС створює додаткові проблеми для розроблювачів ОС, тому що в цьому випадку набагато складніше забезпечити погоджений доступ окремих процесів до загальних системних таблиць, виключити ефект змагань та інші небажані наслідки асинхронного виконання завдань.

Інші вимоги пред'являються до операційних систем кластерів. Кластер - слабко зв'язана сукупність декількох обчислювальних систем, що працюють спільно для виконання загальних додатків, і представляються користувачеві єдиною системою. Поряд зі спеціальною апаратурою для функціонування кластерних систем необхідна й програмна підтримка з боку операційної системи, що зводиться в основному до синхронізації доступу до поділюваних ресурсів, виявленню відмов і динамічної реконфігурації системи.

Поряд з ОС, орієнтованими на цілком визначений тип апаратної платформи, існують операційні системи, спеціально розроблені таким чином, щоб вони могли бути легко перенесені з комп'ютера одного типу на комп'ютер іншого типу, так звані мобільні ОС. Найбільш яскравим прикладом такої ОС є популярна система UNIX. У цих системах апаратно-залежні місця ретельно локалізовані, так що при переносі системи на нову платформу переписуються тільки вони. Засобом, що полегшує перенос іншої частини ОС, є написання її машинно-незалежною мовою.

Багатозадачні ОС поділяються на три класи відповідно до використаних при їхній розробці критеріїв ефективності: системи пакетної обробки, системи поділу часу та системи реального часу.

Системи пакетної обробки призначалися для рішення завдань в основному обчислювального характеру, не потребуючого швидкого одержання результатів. Головною метою й критерієм ефективності систем пакетної обробки є максимальна пропускна здатність, тобто рішення максимального числа завдань в одиницю часу. Для досягнення цієї мети в системах пакетної обробки використовується наступна схема

функціонування: на початку роботи формується пакет завдань, кожне завдання містить вимоги до системних ресурсів; із цього пакета завдань формується мультипрограмна суміш, тобто безліч одночасно виконуваних завдань. Для одночасного виконання вибираються завдання, що пред'являють вимоги, що відрізняються, до ресурсів, так, щоб забезпечувалося збалансоване завантаження всіх пристроїв обчислювальної машини. Таким чином, вибір нового завдання з пакета завдань залежить від внутрішньої ситуації, що складається в системі, тобто вибирається "вигідне" завдання. Отже, у таких ОС неможливо гарантувати виконання того або іншого завдання протягом певного періоду часу. У системах пакетної обробки перемикання процесора з виконання одного завдання на виконання іншого відбувається тільки у випадку, якщо активне завдання саме відмовляється від процесора, наприклад, через необхідність виконати операцію вводу-виводу. Тому одне завдання може надовго зайняти процесор, що унеможливлює виконання інтерактивних завдань. Таким чином, взаємодія користувача з обчислювальною машиною, на якій встановлена система пакетної обробки, зводиться до того, що він приносить завдання, віддає його диспетчерові-операторові, а наприкінці дня після виконання всього пакета завдань одержує результат. Очевидно, що такий порядок знижує ефективність роботи користувача.

Системи поділу часу покликані виправити основний недолік систем пакетної обробки - ізоляцію користувача-програміста від процесу виконання його завдань. Кожному користувачеві системи поділу часу надається термінал, з якого він може вести діалог зі своєю програмою. Так як в системах поділу часу кожному завданню виділяється тільки квант процесорного часу, жодне завдання не займає процесор надовго, і час відповіді виявляється прийнятним. Якщо квант обраний досить невеликим, то у всіх користувачів, що одночасно працюють на одній і тій же машині, складається враження, що кожний з них одноосібно використовує машину. Ясно, що системи поділу часу мають меншу пропускну здатність, ніж

системи пакетної обробки, тому що на виконання приймається кожне запущена користувачем завдання, а не те, котре "вигідне" системі, і, крім того, є накладні витрати обчислювальної потужності на більш часте перемикання процесора із завдання на завдання. Критерієм ефективності систем поділу часу є не максимальна пропускна здатність, а зручність і ефективність роботи користувача.

Системи реального часу застосовуються для керування різними технічними об'єктами, такими, наприклад, як верстат, супутник, наукова експериментальна установка або технологічні процеси, такими, як гальванічна лінія, доменний процес і т.п. У всіх цих випадках існує гранично припустимий час, протягом якого повинна бути виконана та або інша програма, що управляє об'єктом, у противному випадку може відбутися аварія: супутник вийде із зони видимості, експериментальні дані, що надходять із датчиків, будуть загублені, товщина гальванічного покриття не буде відповідати нормі. Таким чином, критерієм ефективності для систем реального часу є їхня здатність витримувати заздалегідь задані інтервали часу між запуском програми й одержанням результату (керуючого впливу). Цей час називається часом реакції системи, а відповідна властивість системи - реактивністю. Для цих систем мультипрограмна суміш являє собою фіксований набір заздалегідь розроблених програм, а вибір програми на виконання здійснюється виходячи з поточного стану об'єкта або відповідно до розкладу планових робіт.

Деякі операційні системи можуть сполучати в собі властивості систем різних типів, наприклад, частина завдань може виконуватися в режимі пакетної обробки, а частина - у режимі реального часу або в режимі поділу часу. У таких випадках режим пакетної обробки часто називають фоновим режимом.

Сучасні ОС в переважній більшості є багатозадачними і багатокоритувацькими системами реального часу, які орієнтовані на роботу на ПК і застосовуються для вирішення різноманітних задач.

1.1.2 Взаємоблокування процесів багатозадачних ОС для ПК

При використанні персональних комп'ютерів значна увага приділяється питанням багатозадачності, що в свою чергу спонукає до ефективнішого вирішення проблем пов’язаних з цим.

Визначення 1.1. Багатозадачність (англ. *multitasking*) — властивість операційної системи або середовища програмування, забезпечувати можливість паралельної (або псевдопаралельної) обробки декількох процесів [2-9]. Дійсна багатозадачність операційної системи можлива тільки в розподілених обчислювальних системах.

Визначення 1.2. Процес – послідовність операцій при виконанні програми, що є наборами байтів, які інтерпретуються центральним процесором як машинні інструкції, дані та стекові структури (абстрактне поняття, що описує роботу програми) [2-9,15,16,20-24].

Процес – це система дій, що реалізує певну функцію в обчислювальній системі й оформлена так, що керуюча програма обчислювальної системи може перерозподіляти ресурси цієї системи з метою забезпечення мультипрограмування [2,6,9,16,20].

Визначення 1.3. Ресурс обчислювальної системи – засіб обчислювальної системи, що може бути виділений процесу обробки даних на певний інтервал часу [2-9,15,16,20-24]. Основними ресурсами обчислювальної системи є: процесори; області основної пам'яті; набори даних; периферійні пристрої; програми.

Прості багатозадачні середовища забезпечують "чистий" розподіл ресурсів, коли за кожним завданням закріплюється певна ділянка пам'яті, і завдання активізується в строго певні інтервали часу.

Більш складніші багатозадачні системи проводять розподіл ресурсів динамічно, коли завдання стартує в пам'яті або залишає пам'ять, залежно від його пріоритету й від стратегії системи. Таке багатозадачне середовище має наступні особливості [2,6,10]:

– кожне завдання має свій пріоритет, відповідно до якого одержує час і пам'ять;

– система організує черги завдань так, щоб всі завдання одержали ресурси, залежно від пріоритетів і стратегії системи;

– система організує обробку переривань, за якими завдання можуть активуватися, деактивуватися й видалятися;

– по завершенні призначеного кванта часу завдання може тимчасово вивантажуватися з пам'яті, віддаючи ресурси іншим завданням, а потім через визначений системою час, відновлюватися в пам'яті;

– система забезпечує захист пам'яті від несанкціонованого втручання інших завдань;

– система розпізнає збої й блокування окремих завдань і припиняє їх;

– система вирішує конфлікти доступу до ресурсів і пристроїв, не допускаючи тупикових ситуацій, загального блокування процесів в результаті очікування заблокованих ресурсів;

– система гарантує кожному завданню, що рано чи пізно воно буде активоване;

– система обробляє запити реального часу;

– система забезпечує взаємодію між процесами.

Основними труднощами при реалізації багатозадачного середовища є забезпечення його надійності, виражене в захисті пам'яті, обробці збоїв і переривань, запобіганні блокувань процесів і тупикових ситуацій [10-13].

Крім надійності, багатозадачне середовище повинно бути ефективним. Витрати ресурсів на його підтримку не повинні заважати

процесам виконуватись, сповільнювати їхню роботу, різко обмежувати використання пам'яті.

При паралельному виконанні задач можуть виникати такі ситуації, при яких два і більше процеси весь час знаходяться в стані блокування, очікуючи спільних ресурсів. Про такі процеси говорять, що вони знаходяться в стані взаємного блокування.

Для виникнення взаємоблокувань в системі є необхідним і достатнім виконання ряду умов, що були сформульовані Кофманом, Елфіком і Шошані в 1970 р., а саме [10]:

1) Умова взаємовиключення (Mutual exclusion). Одночасно використовувати певний ресурс може лише один процес.

2) Умова очікування ресурсів (Hold and wait). Процеси можуть утримувати ресурси, які вже виділені їм, і при цьому можуть запитувати інші ресурси.

3) Умова неперерозподілюваності (No preemtion). Ресурс, що виділений раніше, не може бути примусово відібраний у процесу. Вивільнятись ресурси можуть лише процесом, який їх утримує.

4) Умова кругового (циклічного) очікування (Circular wait). Існує кільцева послідовність процесів, у якій кожний процес очікує доступу до ресурсу, що утримується іншим процесом послідовності.

В даний час розглядаються можливі компромісні рішення з погляду додаткових витрат на включення засобів боротьби з взаємоблокуваннями і очікуваної від цього ефективності. В деяких випадках вартість, яку необхідно витратити на те, щоб зробити систему вільною від взаємоблокувань, дуже висока. Проте, в системах реального часу виникнення ситуації взаємоблокування може призвести до катастрофічних наслідків.

Актуальною проблемою при організації та використанні багатозадачності є виникнення стану взаємоблокування запущених на виконання процесів.

1.2 Моделі, методи та алгоритми розв'язання задачі взаємоблокування

1.2.1 Моделювання взаємоблокувань

У роботі [11] Хольт запропонував змоделювати чотири умови виникнення взаємоблокувань за допомогою напрямлених графів, де використовуються два види вузлів: процеси (позначаються ◯) та ресурси (позначаються □) (рис.1.1).

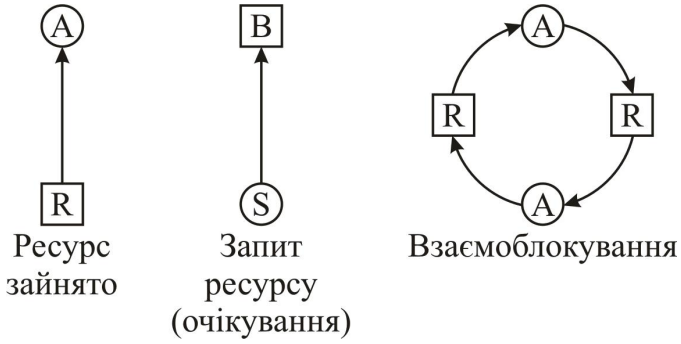


Рисунок 1.1 - Графи розподілу ресурсів

В такій моделі ребро, що з'єднує вузол "ресурс" із вузлом "процес" (напрямлене від вузла "ресурс" до вузла "процес"), означає, що процес надав запит на отримання ресурсу, отримав його і в даний момент використовує. Ребро, що з'єднує вузол "процес" із вузлом "ресурс", означає, що процес в даний момент заблокований і знаходиться у стані очікування ресурсу.

З такої моделі видно, чи виникає в системі взаємоблокування. Наявність циклів свідчить про наявність взаємоблокування.

Розглянемо приклад: нехай в системі є три процеси A, B та C і три ресурси R, S та T. Послідовність запитів і повернень ресурсів для трьох процесів наведені на рис. 1.2(а-в). ОС може запускати на виконання будь-який незаблокований процес в будь-який момент часу. Припустимо, що

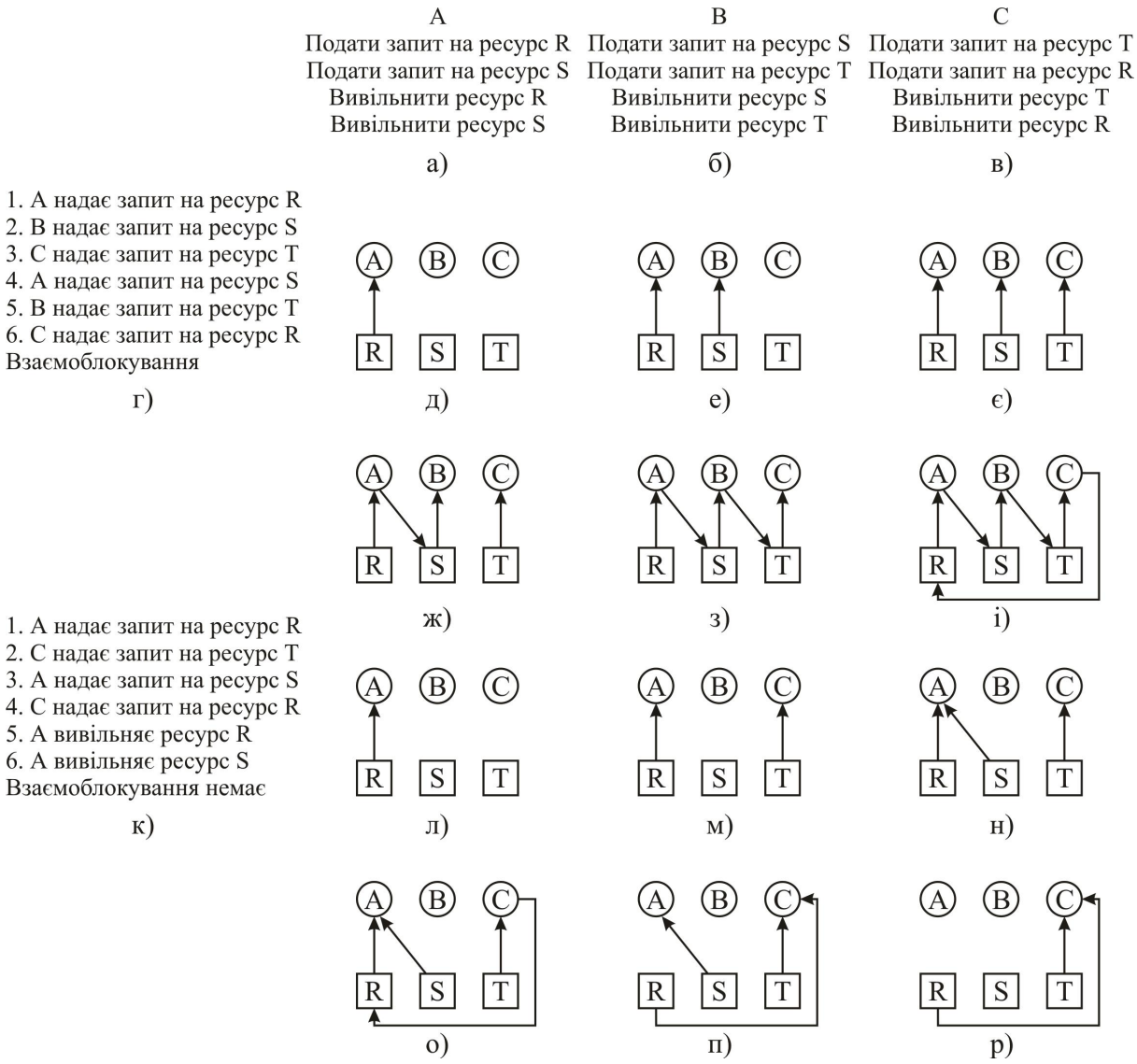


Рисунок 1.2 - Приклад виникнення взаємоблокування і спосіб його уникнення

спочатку буде запущено процес A, який буде виконуватись, поки не завершить свою роботу. Потім буде запущено процес B до його завершення, а потім процес C. Такий порядок не призводить до взаємоблокування. Крім запитів на отримання ресурсів і повернення ресурсів, процеси виконують різноманітні обчислення і введення-виведення даних. Запити ресурсів можуть відбуватись у порядку, вказаному на рис.1.2(г). В результаті такої послідовності дій в системі виникає взаємоблокування (рис.1.2(д-к)). Проте ОС не зобов'язана запускати процеси в певному осо-бливому порядку. Послідовність дій,

представлена на рис.1.2(л) та відображена графами на рис.1.2(м-с), не призведе до виникнення взаємоблокування. Але недоліком такої моделі є неможливість передбачення наперед послідовності запитів та вивільнення ресурсів, а також обчислень та операцій введення-виведення даних.

1.2.2 Основні групи методів та стратегії вирішення задачі взаємоблокування

Відомі методи розв'язування задачі взаємоблокування поділяють на наступні групи [10-13,15, 17-19]:

1) ігнорування проблеми взаємоблокування у випадку дуже низької ймовірності виникнення взаємоблокування;

2) методи що запобігають взаємоблокуванням шляхом створення передумов для невиконання однієї з чотирьох умов настання взаємоблокування;

3) методи виявлення взаємоблокувань, які полягають у тому, щоб завжди задовільняти запити на ресурси, коли це можливо, і періодично перевіряти систему на наявність взаємоблокувань та вирішувати проблему у випадку її настання;

4) методи уникнення взаємоблокування, що полягають у тому, щоб не задовільняти запит на ресурс, якщо його виділення може потенційно спричинити взаємоблокування.

Ігнорування взаємоблокування

Проблема взаємоблокувань може просто бути проігнорована, тобто використаний підхід без стратегії у вирішенні проблеми взаємоблокування. Перевагою цього методу є економія процесорного часу та інших ресурсів, необхідних для виявлення взаємоблокувань. Однак при такому підході відповідальність за виявлення взаємоблокування покладається на кожного оператора, який досліджує заблоковані процеси, чи користувача, що очікує на відповідь від системи.

Для того, щоб ухвалити таке рішення, необхідно оцінити ймовірність виникнення взаємоблокування і порівняти її з ймовірністю відмов іншого апаратного і програмного забезпечення. Розробники ОС в більшості випадків не бажають знижувати продуктивність системи або позбавляти зручності у роботі користувачів за рахунок впровадження складних і дорогих засобів боротьби з взаємоблокуванням.

В будь-якій ОС, що має в ядрі ряд масивів фіксованої розмірності, виникають взаємоблокування, навіть якщо вони не виявлені. Таблиця відкритих файлів, таблиця процесів та ін. є обмеженим ресурсом. Заповнення всіх записів таблиці процесів може привести до того, що черговий запит на створення процесу може бути відхилений. При несприятливому збігу обставин декілька процесів можуть видати такий запит одночасно і при цьому потрапити у взаємоблокування.

В більшості поширених ОС (Unix, Windows і ін.) розробники проігнорували дану проблему [5,20,21,23,37] припускаючи, що низька ймовірність виникнення взаємоблокування краще, ніж правила, що примушують користувачів обмежувати число процесів, відкритих файлів і тому подібне.

Запобігання взаємоблокуванню

У більшості методах даної групи пропонується забезпечити кожен процес ресурсами, які йому потрібні, до початку будь-яких дій. Взаємоблокування може бути уникнуто кількома шляхами: запитом усіх необхідних йому ресурсів, запобіганням утриманню ресурсів, впорядкуванням ресурсів.

Найпростіший спосіб запобігти взаємоблокуванню – це уникнення паралелізму. Але це призводить до дуже примітивного використання ресурсів і зниження продуктивності системи. Інший підхід вимагає, щоб усі потреби у ресурсах були задоволені до початку опрацювання. Такі схеми непридатні, коли утримання ресурсів може відбуватися впродовж тривалого часу, але працюють добре для процесів, які виконують

одноразові активні дії, наприклад, введення-виведення, коли ресурси можуть звільнитися після кожного використання. Для процесів, запити яких постійно змінюються, цей метод непрактичний. Наприклад, коли процесу потрібно більше основної пам’яті, ніж є доступно зараз, він стає блокованим. Так, процес поміщається в додатковий запам’ятовуючий пристрій, так як його пам'ять використовується активним процесом. Заблокований процес вийде зі свого стану лише коли необхідна кількість пам’яті буде доступною.

Розглянемо основні алгоритми запобігання взаємоблокуванню.

Одноразовий алгоритм [2-6]. Процес P подає запит на ресурси R, … Rn.

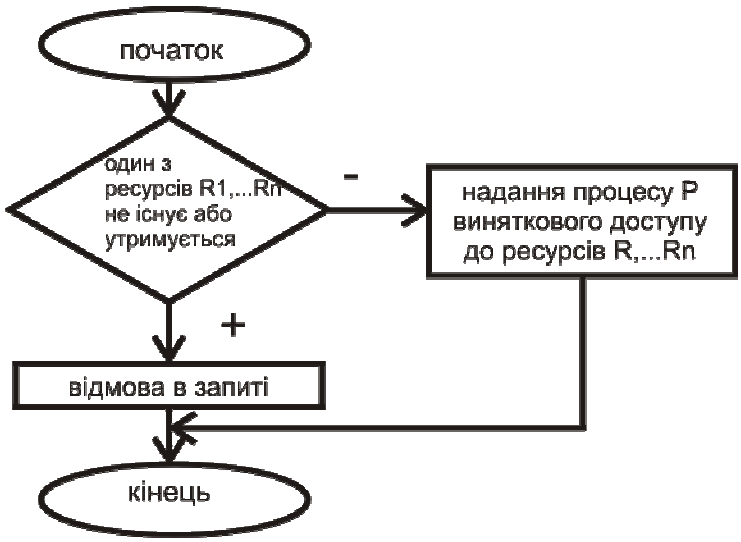


Рисунок 1.3 - Одноразовий алгоритм запобігання взаємоблокуванню

Ієрархічний алгоритм [2-6]. Процес P подає запит на ресурс R.

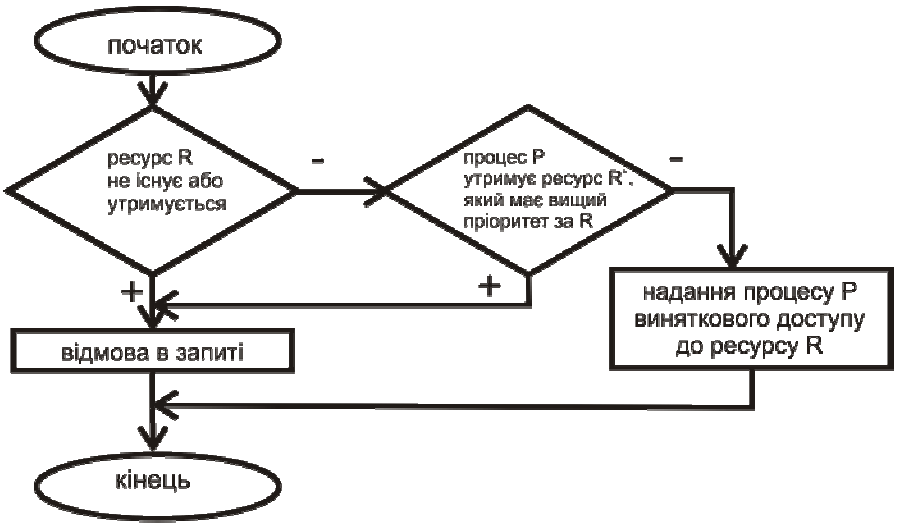


Рисунок 1.4 - Ієрархічний алгоритм запобігання взаємоблокуванню

Ієрархічний алгоритм з очікуванням [2-6]. Процес P подає запит на ресурс R.

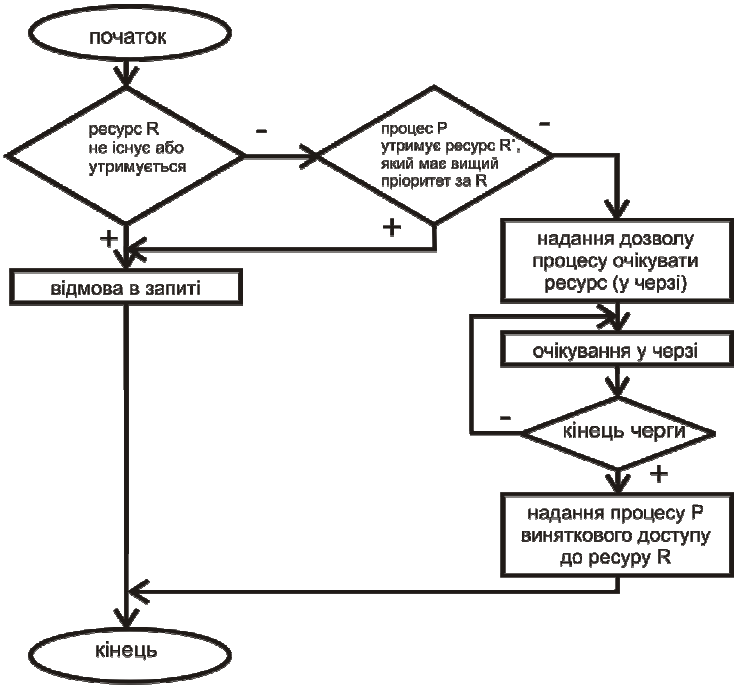


Рисунок 1.5 - Ієрархічний алгоритм з очікуванням

Виявлення взаємоблокувань

Ці методи розраховують, що усі запити на ресурси будуть задоволені. Алгоритм, що періодично викликається, аналізує усі стани ресурсів та запити на них, щоб виявити, чи існує взаємоблокування

процесів або ресурсів. Якщо взаємоблокування виявлене, то система повинна вирішити проблему, якомога безпечніше, взявши ресурси під контроль перед процесами, які пов’язані з ними.

Оскільки ніяких дій не відбувається поки взаємоблокування не виникне, ресурси можуть утримуватись заблокованими процесами протягом великого проміжку часу. Інколи ефективне використання механізмів вирішення проблеми взаємоблокування є складним, наприклад, коли відбирання ресурсів, таких, як стрічковий накопичувач, може призвести до неприпустимих наслідків. Проте механізми вирішення проблем взаємоблокування мають багато переваг, так як застосовуються періодично і з мінімальним набором дій у випадку відсутності взаємоблокування.

У випадку виявлення взаємоблокування існує декілька варіантів його усунення [2-6,16-24]:

1) Відновлення системи за допомогою примусового вивантаження ресурсу, що вимагає втручання користувача.

2) Відновлення через відкат. Стан процесів записується в контрольних точках, і у випадку тупика можна зробити відкат процесу у попередній стан, після чого він продовжить роботу знову із цієї точки.

3) Відновлення шляхом знищення одного із процесів

Проте у системах реального часу дані підходи до усунення взаємоблокування є непридатними, оскільки для таких систем є критичним час завершення або початку виконання процесу [2-6,9].

Схема виявлення описана вище охоплює не тільки виявлення взаємоблокувань в процесі виконання, а й потенційні небезпеки.

Уникнення взаємоблокувань

У методах уникнення взаємоблокувань запит на ресурс дозволяється лише коли існує хоча б один безпечний шлях виконання для всіх. Одна базова схема, пов’язана з алгоритмом банкіра, використовує кратні кількості певного ресурсу, з вимогою процесів визначати свої загальні

потреби в початковий момент часу. До того ж, кожен процес подає запит чи звільняє множину ресурсів по одному за один раз. Алгоритм дає відмову процесам, потреби яких перевищують наявні ресурси.

Хеберман розробив стратегію максимального дозволу [5,6,16] для того, щоб контролювати майбутні потреби процесів в ресурсах. Узагальнення алгоритму банкіра є практичним прикладом уникнення, але вимагає наявності певного об'єму інформації. Уникнення взаємоблокування досягається перевіркою усіх виділень ресурсів і допуском лише тих, які є безпечними. Якщо процес, що виконує це виділення, може виконуватися до свого завершення і відпускати ресурси, які ним утримуються, тоді всі інші процеси в системі можуть бути завершені.

Порівняння підходів

В таблиці 1.1 представлено порівняльну характеристику підходів вирішення проблеми взаємоблокування.

Таблиця 1.1 - Порівняння методів вирішення проблеми взаємоблокування

Метод	Політика розподілу ресурсів	Відмінності схем	Основні переваги	Основні недоліки
1	2	3	4	5
Виявлення взаємоблокувань	Усі запити на ресурси задовольняються	Періодичне застосування для виявлення взаємоблокування	Ніколи не затримує запуск процесів. Сприяє оперативному вирішенню проблеми	Можуть бути присутні втрати даних

Продовження таблиці 1.1

1	2	3	4	5
Запобігання взаємоблокуванням	Контроль виділення ресурсів	Запит усіх ресурсів одразу. Приорітетність. Впорядкування ресурсів	Працює добре для процесів які проявляють одноразову активність Ефективний. коли застосовується до ресурсів, стан яких може бути легко збережений та відновлений Можливість застосування при перевірках під час компіляції Не потрібно перевірок під час виконання через системну архітектуру	Неефективний, затримує запуск процесів. Попереджає частіше ніж необхідно. Можливий циклічний перезапуск процесу. Відмовляє зростаючим запитам на ресурси

Кінець таблиці 1.1

1	2	3	4	5
Уникнення взаємобло-кувань	Займає проміжну позицію між виявленням та запобіганням	Проводить маніпуляції щоб знайти хоча б один безпечний шлях	Немає необхідності в негайному застосуванні	Мають бути відомі майбутні потреби в ресурсах Процеси можуть бути заблоковані довгий час

Як видно із таблиці 1.1, кожна група методів має свої переваги та недоліки. Найбільш придатними для використання є методи, які запобігають та уникають взаємоблокування, оскільки вони мають найменші втрати продуктивності системи.

Проте найкращим вирішенням проблеми взаємоблокування є комбінація відомих методів, а саме уникнення та запобігання взаємоблокуванню, оскільки при цьому підвищується надійність роботи ОС.

1.2.3 Методи та алгоритми вирішення задачі взаємоблокування для ОС ПК шляхом його уникнення та запобігання

В процесі експлуатації персональних комп'ютерів (ПК) досить часто виникає ситуація блокування процесів, що виконуються на них [15]. Причинами такого явища є помилки у програмному коді ПЗ, невідповідність ПЗ його специфікації. Наслідком такої ситуації є вихід з ладу апаратних складових, некоректне звертання програмного забезпечення (ПЗ) до пам'яті, конфліктні ситуації між різним ПЗ.

Для визначення і усунення причин і наслідків блокування процесів у комп'ютерній системі використовують різноманітні методи і засоби, а

саме: семафори, монітори, м'ютекси [2-6], спеціалізовані діагностичні програми для перевірки програмної та апаратної цілісності ПК, тестери для перевірки справності апаратної частини [15]. Проте на практиці їх не завжди доцільно застосовувати для вирішення задачі блокування.

Частковим випадком блокування програм на ПК є виникнення взаємоблокування. Дане питання досліджувалось Дейкстрою, Хоаром, Брінч-Хансеном, Деккером, Петерсеном, Коффманом, Хольтом [2-9]. Розроблено багато методів та алгоритмів для уникнення взаємоблокувань процесів. Проте частина з них не може бути реалізованою в сучасних операційних системах і носить теоретичний характер [2-6]. Інша частина при реалізації стає досить громіздкою в структурі операційної системи. Тому розробники більшості ОС не включали відомі алгоритми уникнення взаємоблокувань процесів. За умови виконання в системі невеликої кількості процесів відсутність таких засобів була допустима. Однак, стрімкий розвиток апаратних засобів ПК, зростання об’єму та складності ПЗ та розв’язуванням на ПК відповідальних задач вимагає недопущення виникнення стану взаємоблокування, що в свою чергу вимагає розробки нових підходів до вирішення задачі взаємоблокування процесів.

Як відомо, основним ресурсом обчислювальної машини є її процесор (або процесори) [2-9, 15, 16, 20-24]. У кожний момент часу один процесор може виконувати тільки один процес. Однією з основних задач будь-якої багатокористувацької і багатозадачної операційної системи є організація планування процесів, оскільки вона безпосередньо впливає на ефективність функціонування обчислювальної машини. Існує декілька варіантів визначення процесу. Процес (або по-іншому, завдання) - абстракція, що описує програму, яка виконується. Для операційної системи процес являє собою одиницю роботи, заявку на використання системних ресурсів. Тому кожному процесу мають бути виділені наступні системні ресурси: процесор, пам'ять, доступ до пристроїв вводу-виводу, файли.

Процес складається з коду програми, лічильника інструкцій, стеку, розділу даних. Для кожного процесу створюється свій блок керування [2-9, 16, 20-24], що поміщається в системну таблицю процесів, що знаходяться у ядрі. Ця таблиця містить масив структур блоків керування процесами. У кожному блоці містяться дані: слово стану процесу; пріоритет; величина кванта часу, виділеного системним планувальником; ступінь використання системним процесором; ознака диспетчеризації; ідентифікатор користувача, якому належить процес; ефективний ідентифікатор користувача; реальний і ефективний ідентифікатори групи; група процесу; ідентифікатор процесу й ідентифікатор батьківського процесу; розмір образа, що розташовується в області підкачування; розмір сегментів коду й даних; масив сигналів, що очікують обробки.

Протягом існування процесу його виконання може бути багаторазово перервано й продовжено, тобто процеси змінюють свій стан [2-9, 16, 20-24]. Для того, щоб відновити виконання процесу, необхідно відновити стан його операційного середовища. Стан операційного середовища відображається контекстом процесу – станом регістрів і програмного лічильника, режимом роботи процесора, покажчиками на відкриті файли, інформацією про незавершені операції вводу-виводу, кодами помилок виконуваних даним процесом системних викликів.

В багатозадачній комп'ютерній системі процеси приймають наступні стани [2-6] (рис.1.6), а саме: стан “створений”, стан “очікуючий”, стан “виконуваний”, стан “заблокований”, стан “завершений”. У системах із віртуальною пам’яттю додаються ще два стани: стан “вивантажений та очікуючий” і стан “вивантажений та заблокований”.

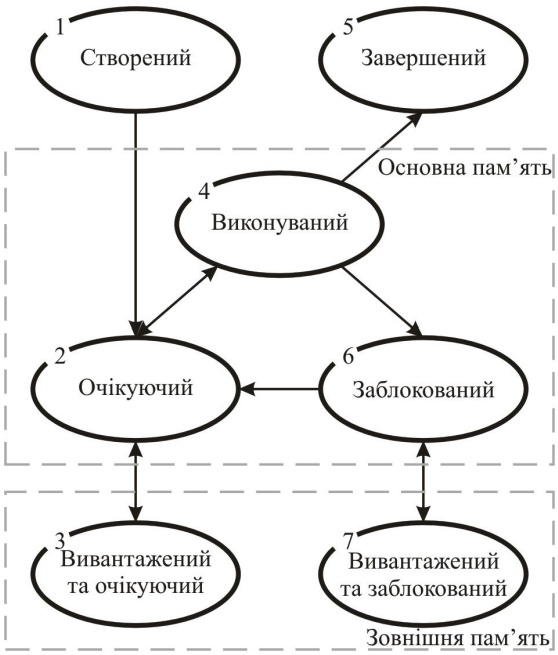


Рисунок 1.6 - Діаграма станів процесу

Подамо переходи процесу між станами за допомогою таблиці (матриця суміжності графа) (див. табл.1.2).

Таблиця 1.2 - Матрична форма діаграми станів процесу

Стан	1	2	3	4	5	6	7
1		1					
2			1	1			
3		1					
4					1	1	
5							
6		1					1
7						1	

Серед перерахованих станів найбільш критичним є стан "заблокований", у якому процес не може виконуватись, оскільки очікує отримання певного ресурсу. Якщо процес "заблокується" на ресурсі (файлі, семафорі, пристрої тощо), він буде усунутий з процесора (так як процес не може продовжувати виконання) і переведений в блокований стан. Процес буде залишатися "заблокований" доки відповідний ресурс не стане

доступний. Про розблокування ресурсу заблокований процес повідомляє операційна система (про доступність ресурсу сама операційна система повідомляється з допомогою переривання). Як тільки операційна система дізнається, що процес розблокований, він переводиться в стан "готовий", з якого він може бути переведений в стан "виконуваний", в якому він зможе використати заново доступний ресурс.

На рис. 1.7а представлено часову діаграму виконання процесів для однопроцесорної багатозадачної ОС, на рис. 1.7б – для багатопроцесорної багатозадачної ОС.

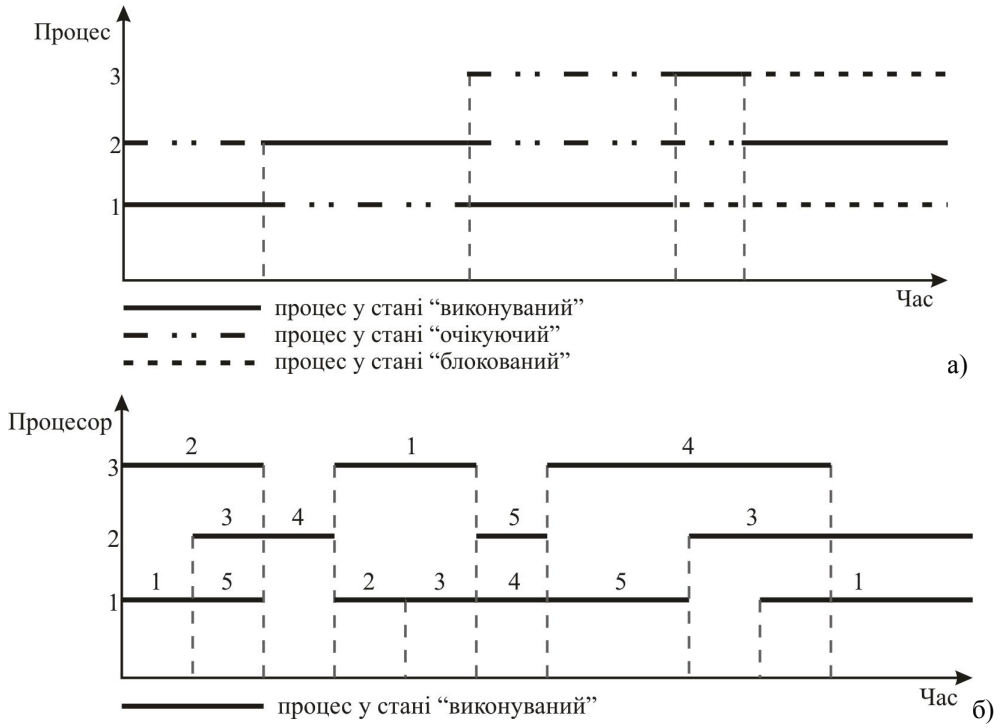


Рисунок 1.7 - Часова діаграма виконання процесів:

а) для однопроцесорної багатозадачної ОС

б) для багатопроцесорної багатозадачної ОС

Частковим випадком виконання процесів є стан взаємоблокування: коли два процеси намагаються захопити спільні ресурси і блокують виконання один одного; коли два процеси взаємодіють між собою і для свого продовження вимагають результати роботи один одного. Для

вирішення задачі взаємоблокування існують різноманітні методи і алгоритми [2-24]. Проте вони мають ряд недоліків.

Метод уникнення взаємоблокувань “заборона на переривання” полягає у забороні усіх переривань при входженні процесу у критичну ділянку і дозволі усіх переривань при виході із неї [3,5-7,10]. Проблемою при такому підході є ймовірність настання ситуації, коли процес у результаті деякого збою у роботі не відновлює дозвіл на переривання при виході із критичної ділянки. Це призводить до блокування роботи усіх процесів операційної системи.

Метод уникнення взаємоблокувань “змінні блокування” полягає у використанні деякої загальної змінної із початком значенням "0", яка сигналізує про те, чи знаходиться певний процес у критичній ділянці [3,5-7,10]. Проблемою даного методу є те, що перш ніж один процес змінить значення змінної на 1, інший процес може отримати керування, виконати перевірку цієї змінної і увійти у критичну ділянку. Таким чином два процеси можуть одночасно опинитись у критичній ділянці.

Метод уникнення взаємоблокувань “строге чередування” полягає у почерговому входженні кожного процесу у критичну ділянку, що контролюється додатковою змінною [3,5-7,10]. Проблемою даного методу є те, що він не враховує швидкість виконання процесів і при наявності повільних процесів настає ситуація коли більш швидкі процеси змушені довго очікувати їхнього завершення.

Алгоритм Деккера [3,5-7,10] для уникнення взаємоблокувань полягає у використанні двох змінних: однієї, яка містить ім'я процесу, що має право на вхід у критичну ділянку в даний момент; і вектора, який містить імена процесів, що потребують входження у критичну ділянку. Недоліком даного алгоритму є складність його реалізації у випадку наявності більш ніж для 2 процесів. Також він вимагає входження процесів у критичну ділянку у заданому порядку, що не враховує швидкість виконання

процесів. Тому при наявності повільних процесів настає ситуація коли більш швидкі процеси змушені довго очікувати їхнього завершення.

Алгоритм Петерсона [3,5-7,10] для уникнення взаємоблокувань полягає також у використанні двох змінних: однієї, яка містить ім'я процесу, що має право на вхід у критичну ділянку в даний момент; і вектора, який містить імена процесів, що потребують входження у критичну ділянку. Недоліком даного алгоритму є складність його реалізації у випадку наявності більш ніж для 2 процесів.

Метод уникнення взаємоблокувань “семафори” полягає у використанні спеціальних змінних, над якими визначені операції ініціалізації початковим значенням, збільшення значення на 1, зменшення значення на 1 [3,5-7,10]. Проблемою даного методу є неможливість передбачити якими розділюваними ресурсами буде володіти ОС і скільки потрібно буде використати семафорів. Також є ймовірність виникнення ситуації взаємоблокування із-за неправильного використання семафорів.

Метод уникнення взаємоблокувань “м'ютекси” полягає у використанні спеціальних змінних, які можуть знаходитись у 2 станах: блокованому і неблокованому [3,5-7,10]. Але на відміну від семафорів у м'ютексів відсутнє активне очікування процесів перед входженням у критичну ділянку. Недоліком даного методу є необхідність використання спеціалізованої команди процесора TSL.

Метод вирішення уникнення взаємоблокувань “монітори” полягає у використанні набору процедур, змінних і інших структур даних, що об’єднані у особливий модуль [3,5-7,10]. Проблемою даного методу є те, що монітори є структурними компонентами компілятора, а сучасні компілятори не містять таких компонент.

Перераховані методи (табл.1.3) не орієнтовані на реалізацію в сучасних операційних системах персональних комп’ютерів. Також вони не дозволяють передбачати виникнення ситуацій взаємоблокування для трьох і більше процесів.

Таблиця 1.3. - Недоліки існуючих методів уникнення взаємоблокувань

№ п/п	Назва методу	Реалізація	Недоліки
1.	Заборона на переривання	Апаратна	Блокування роботи ОС, активне очікування
2.	Змінні блокування	Програмна	Часткове вирішення задачі блокування, активне очікування
3.	Строге чередування	Програмна	Зниження продуктивності роботи обчислювальної системи, активне очікування
4.	Алгоритм Деккера	Програмна	Складність програмної реалізації, активне очікування
5.	Алгоритм Петерсона	Програмна	Складність програмної реалізації для багатьох процесів, активне очікування
6.	Семафори	Програмна	Можливість блокування через порушення порядку використання, активне очікування
7.	М'ютекси	Програмно-апаратна	Наявність спеціалізованої команди процесора
8.	Монітори	Програмна	Необхідність підтримки на рівні компіляторів

Як видно із таблиці 1.3, основними недоліками розглянутих методів є складність програмної реалізації для багатьох процесів, зниження продуктивності системи, наявність циклів активного очікування, блокування роботи ОС, необхідність наявності та використання

спеціалізованої команди процесора. Серед перерахованих недоліків найбільш критичними є блокування роботи ОС при певних умовах, необхідність спеціалізованої команди процесора та складність програмної реалізації для багатьох процесів.

1.3 Стратегії вирішення задачі взаємоблокування

Аналіз відомих підходів до вирішення задачі взаємоблокування процесів показав, що вони мають ряд суттєвих недоліків, які роблять їх непридатними для використання у сучасних ОС. Одним із підходів до вирішення такої задачі є передбачення входження процесів у стан взаємоблокування та недопущення цього.

Сучасний етап розвитку теорії штучного інтелекту (ШІ) дозволяє застосовувати на практиці її компоненти у багатьох галузях, де необхідно швидко вирішувати важкоформалізовані задачі. При цьому використовуються як системи для підтримки прийняття рішень (СППР), у яких останнє слово лишається за оператором, так і системи в автономному режимі, коли система самостійно приймає остаточне рішення.

Для прогнозування стану об'єктів найефективнішим є використання таких компонентів теорії ШІ, як [44, 45]:

- штучні нейронні мережі (ШНМ);
- штучні імунні системи (ШІС)
- чіткі та нечіткі експертні системи (ЕС);

Розглянемо кожен із компонентів ШІ у ракурсі його використання в системах прогнозування стану об'єктів.

Штучні нейронні мережі знайшли застосування в самих різних галузях людської діяльності – медицина, економіка, техніка. ШНМ використовуються при вирішенні задач керування складними об‘єктами, оптимізації, розпізнавання образів, ідентифікації, прогнозування та інших [68, 79, 80, 81, 82]. В технічній діагностиці ШНМ використовуються для

визначення технічного стану об‘єкта діагностування, генерації тестових векторів та інше [83, 84].

ШНМ будуються за принципами організації та функціонування їх біологічних аналогів – клітин мозку. Вони є розподіленими та паралельними системами здатними до адаптивного навчання шляхом аналізу позитивних та негативних впливів [85 - 90]. Елементарним перетворювачем в ШНМ є штучний нейрон [91 – 97]. Штучні нейрони в свою чергу об‘єднуються в мережі з певною структурою.

ШНМ реалізуються у вигляді програмного, апаратного чи програмно-апаратного комплексу. Програмний спосіб реалізації полягає у використанні спеціалізованих пакетів прикладних програм (нейропакетів), що дозволяють моделювати роботу ШНМ на машинах фон Нейманівського типу [98, 99]. Нейропакети можуть бути окремими оболонками (BraineMaker фірми California Scientific Software, NeuroSolutions фірми NeuroDimension Inc), надбудовами над електронними таблицями (BRAINCEL for Excel, 1-2-3 фірми Promised Lahd Technologies) чи входити у склад спеціалізованих математичних пакетів (Neural Network PC Toolbox фірми The MathWorks Inc. для пакета MATLAB).

Програмно-апаратний підхід до реалізації ШНМ полягає у використанні потужних співпроцесорів. Використання нейронних співпроцесорів дозволяє значно збільшувати розмірності моделей та їх швидкодію [66, 99].

Суть апаратного підходу полягає у фізичній реалізації ШНМ на елементній базі у вигляді паралельних нейроструктур, що відповідає визначеній нейропарадигмі. Практичне втілення нейросистем стримувалось відсутністю елементної бази [100, 101, 102, 103], але з появою НВІС та оптичної елементної бази з‘явилися можливості для вдалих реалізацій.

Суттєвою перевагою ШНМ перед експертними системами та нечіткою логікою є здатність до самонавчання, адаптивні та узагальнюючі властивості.

Штучні імунні системи – це адаптивні системи, що побудовані на основі властивостей і принципів функціонування природних імунних систем живих організмів, за якими ведуться спостереження.

Імунна система організму являє собою складну адаптивну структуру, що ефективно використовує різні механізми захисту від зовнішніх патогенів. Основна задача імунної системи полягає в розпізнаванні клітин (або молекул) організму й класифікації їх як своїх або чужих. Чужорідні клітини, що виявляються, служать сигналом для активації захисного механізму відповідного типу.

Моделі, засновані на принципах функціонування систем імунітету, застосовуються в різних галузях науки й техніки. Сфера їхнього застосування включає наступні області (але не обмежується ними) [Дасгупта 2006]:

- методи обчислень;
- когнітивні моделі;
- штучні імунні системи для розпізнавання образів;
- методи виявлення аномалій і несправностей;
- мультиагентні системи;
- моделі самоорганізації;
- моделі колективного інтелекту;
- системи пошуку й оптимізації;
- моделі автономних розподілених систем;
- моделі штучного життя;
- системи комп'ютерної й інтернет-безпеки;
- моделі систем, що навчаються;
- методи добування інформації;
- штучні імунні системи для виявлення підробок;

- методи обробки сигналів і зображень.

З погляду організації обробки даних імунна система - це високопаралельна структура. У ній реалізовані механізми навчання, пам'яті й асоціативного пошуку для розв'язання задач розпізнавання й класифікації. Зокрема, імунна система здатна навчатися розпізнаванню важливих структур (антигенних пептидів); запам'ятовуванню тих структур, що вже зустрічалися, і використанню законів комбінаторики в рамках генних бібліотек для ефективної генерації детекторів структур (варіабельних ділянок молекул антитіл), взаємодіючих із зовнішніми антигенами й власними клітинами організму. При цьому реакція на антиген відбувається не тільки на рівні окремих одиниць, що розпізнають, але й на системному рівні шляхом взаємного розпізнавання клонів лімфоцитів у реакціях антиген-антитіло. Таким чином, поведінка імунної системи визначається всією сукупністю локальних мережних взаємодій.

Для пояснення імунологічних механізмів існують різні теорії й математичні моделі. Також є зростаюче число комп'ютерних моделей для імітації динаміки різних компонентів імунної системи і її поведінки в цілому. Ці підходи включають моделі, сформульовані у вигляді систем диференціальних і стохастичних рівнянь, клітково-автоматні моделі, моделі простору конфігурацій і інші. Разом з тим природна імунна система служить джерелом нових ідей для розвитку інтелектуальних методів розв'язання складних задач, але робіт у цій області поки небагато. Необхідно проводити більше досліджень, зокрема для вивчення механізмів обробки інформації в імунній системі, що може мати велике практичне значення.

На жаль, у цей час існує лише невелике число обчислювальних моделей, заснованих на принципах роботи імунної системи.

ШІС володіють наступними обчислювальними можливостями:

- Навчання
- Запам'ятовування

- Розпізнавання
- Виділення особливостей
- Різноманіття (комбінаторні механізми)
- Розподілена обробка даних (розподілений пошук)
- Адаптація
- Саморегулювання
- Розподіл клітин на власні і чужі
- Динамічний захист
- Ймовірнісне виявлення

Експертні системи – це системи, що ґрунтуються на знаннях та орієнтовані на розв‘язування задач у вузькій предметній області [45, 46, 47, 48].

Інтерес, що викликають експертні системи при розв‘язуванні прикладних задач пов‘язаний з їх властивостями [45, 49, 50]. ЕС орієнтовані на використання в неформалізованих областях, де використання обчислювальної техніки раніше вважалося неможливим. ЕС дають можливість спеціалістам не знайомим з програмуванням, самостійно створювати додатки за допомогою комп‘ютерної техніки. Використовуючи ЕС в процесі розв‘язання прикладних задач, досягаються практичні результати, що переважають можливості інженерів-експертів, які не використовують ЕС. До ЕС підвищений комерційний інтерес, оскільки інструментальні засоби ЕС легко інтегруються з іншими інформаційними технологіями і засобами, можуть розроблятися з дотриманням стандартів, котрі забезпечують переміщуваність. Програмні додатки можуть відпрацьовуватись в системах типу клієнт-сервер та інше.

ЕС складаються з відповідних основних частин: бази знань, робочої пам‘яті, діалогового компонента, інтерпретатора та компонента набуття знань [51 - 57].

На сьогодні існують різнотипні інструментальні засоби розробки ЕС, а саме: окремі готові компоненти (призначені для розробників, що

знаються на програмуванні та здатні сформувати з компонентів єдиний комплекс) [58, 59]; оболонки ЕС загального призначення (містять усе необхідне для створення повноцінної ЕС крім знань) [60]; спеціалізовані оболонки (можуть містити певні знання про конкретні предметні області) [51, 62].

Основним недоліком ЕС прогнозування є те, що вони дають можливість в явному вигляді реалізовувати лише евристичні методи прогнозування. Решта методів можуть бути закладені у знання та використовуватись лише у неявному вигляді. Іншими суттєвими недоліками є збільшення складності евристичних методів із збільшенням складності об‘єкту прогнозування та відсутність експертів для нових МПП та С, стосовно яких ще не накопичено достатнього досвіду.

Нечітка логіка. Часто виникають ситуації, коли процеси описуються множинами, в яких про частину елементів не можна однозначно сказати належать вони відповідній множині чи ні [65 - 69]. Особливо це проявляється при поданні знань експертів. Існує багато речей, які експерти не можуть описати точними виразами. Що стосується діагностики та прогнозування технічного стану МПП та С, то тут можуть зустрічатися вирази типу: "частково справний", "незначні порушення в роботі", "часті збої" та інше [70, 71]. При такому підході ступінчата характеристична функція чіткої множини замінюється неперервною функцією належності нечіткої множини.

Нечітка множина визначається як набір впорядкованих пар [68, 72]:

$$M = \{\mu_M(x)/x\}\ , \qquad (1.1)$$

де $\mu_M(x)$- характеристична функція належності, що приймає значення з певної цілком впорядкованої множини. Функція належності вказує на ступінь належності елемента x до нечіткої множини M .

Механізм нечітких висновків, що використовується в різного роду експертних та управляючих системах, в основі має базу знань, що формується спеціалістами в предметній області. Логічний висновок

виконується за наступні чотири етапи: *введення нечіткості* (функції належності, визначені на вхідних змінних, застосовуються до їх фактичних значень для визначення ступеня істинності кожної з передумов); *логічний висновок* (обчислене значення істинності для передумов кожного правила застосовується до висновків кожного правила, це призводить до однієї нечіткої множини, яка буде назначена кожній змінній висновку для кожного правила); *композиція* (всі нечіткі підмножини об‘єднуються разом, щоб сформувати одну нечітку підмножину для всіх змінних висновку); *приведення до чіткості* (використовується якщо необхідно перетворити вихідний набір у число) [68, 72, 73, 74].

Системи що базуються на нечітких множинах розроблені та успішно впроваджені в таких галузях, як керування технологічними процесами, керування транспортом, медична та технічна діагностика, фінансовий менеджмент, біржове прогнозування, розпізнавання образів та інше [72, 74, 75]. Практичний досвід розробки систем нечіткого логічного висновку свідчить, що строки та собівартість їх проектування значно менші, ніж при використанні традиційного математичного апарату, при цьому забезпечується необхідний рівень роботоздатності та прозорості моделей.

Системи та моделі на базі нечіткої логіки використовуються для вирішення задач діагностування та прогнозування в різних галузях діяльності людини [74, 75, 76, 77, 78]. Прогнозування у системах на базі нечіткої логіки базується на евристичних методах. Зокрема у [78] запропонований новий автоматизований метод вибору оптимальних експертів для вирішення задач діагностування несправностей та прогнозування у системах на базі нечіткої логіки. Найбільш оптимальним використання систем на базі нечіткої логіки є у галузях, де вихідні данні, або знання про предметну область є нечіткими. В решті випадків використання останніх є менш ефективним.

1.4 Постановка задачі

Вагома частка взаємоблокувань припадає на взаємоблокування процесів, що виконуються в операційних системах (ОС) для персональних комп'ютерів (ПК).

В результаті дослідження відомих методів та алгоритмів уникнення взаємоблокувань процесів в комп’ютерній системі виявлено, що вони не в повному обсязі вирішують поставлену задачу і не всі з них придатні до реалізації у сучасних операційних системах, оскільки мають ряд недоліків (блокування роботи ОС, наявність циклів активного очікування, складність програмної реалізації для багатьох процесів, необхідність використання спеціалізованої команди процесора) і є складними для їх реалізації. Тому більшість сучасних ОС не містять засобів для вирішення проблеми взаємоблокування процесів. Проте масштабність задач, які розв’язуються на персональному комп’ютері, зростає і це вимагає вирішення задачі уникнення взаємоблокування.

Для забезпечення безвідмовної роботи процесів, що виконуються на ПК, необхідно вирішити наступні задачі:

1) провести аналіз процесів та їх взаємодії у сучасних операційних системах як об’єктів діагностування та виявити проблеми, що виникають у процесі їх роботи;

2) провести аналіз відомих методів вирішення задачі взаємоблокування процесів з метою дослідження та систематизації їх недоліків;

3) розробити метод прогнозування стану процесів в ПК, який на відміну від відомих дозволив би уникати взаємоблокування процесів та був простим у реалізації у сучасних ОС;

4) розробити методику та алгоритми визначення процесів, що наближаються до стану взаємоблокування, на основі прогнозування їх стану;

5) реалізувати метод прогнозування стану процесу у вигляді програмних додатків та впровадити їх з метою підвищення безвідмовної роботи процесів, що виконуються на персональних комп’ютерах.

2 МОДЕЛЬ ПРОЦЕСУ ПРОГНОЗУВАННЯ СТАНУ ПРОЦЕСІВ В ПЕРСОНАЛЬНОМУ КОМП'ЮТЕРІ

2.1 Сигнатура процесу як його характеристика

Прогнозування стану процесу в ПК полягає у визначенні моменту наближення процесу до стану взаємоблокування на основі контролю та аналізу поточних показників системи та процесу.

Визначимо параметри процесу, що є необхідними для побудови його сигнатури. Для цього розглянемо множину характеристик процесу, якими він володіє у сучасних операційних системах.

Операційна система створює процес, коли користувач запускає програму на виконання. Процес і програма, що виконується в даному процесі, не ототожнюються. Програмою є машинний код і початкові дані, що містяться у виконуваному файлі на диску або скопійовані ОС для виконання в оперативну пам'ять. Для кожного процесу ОС створює додатковий набір даних, що називається середовищем виконання процесу або контекстом процесу. Дані із цього набору називаються атрибутами процесу [4,5] – ознаками, що відрізняють один процес від іншого. Вони можуть бути як сталими, так і змінюватись протягом життєвого циклу процесу (наприклад, ідентифікатор процесу – сталий параметр, пріоритет процесу – динамічний параметр).

До найважливіших атрибутів процесу віднесемо наступні:

– ідентифікатор процесу (PID – Process IDentifier). Ідентифікатор процесу є цілим додатнім числом. Кожний процес в системі має унікальний ідентифікатор. ОС керує процесами, використовуючи ідентифікатор процесу;

– ідентифікатор батьківського процесу (PPID). Цей атрибут процес отримує під час свого запуску і може використовувати його для отримання інформації про стан батьківського процесу або для передачі йому сигналу;

– реальний і ефективний ідентифікатори користувача і групи (UID,

GID, EUID і EGID). Реальні ідентифікатори співпадають з ідентифікаторами користувача, що запустив процес, і групи, до якої він належить. Реальні ідентифікатори наслідуються дочірнім процесом і не можуть бути змінені. Права доступу процесу до ресурсів ЕОМ визначаються ефективними ідентифікаторами;

– відкриті файли. Крім стандартних файлів вводу, виводу і помилок процес може відкрити інші файли. Дочірній процес наслідує всі файли, відкриті батьківським процесом;

– пріоритет і відносний пріоритет. Пріоритет процесу – змінна величина, яка обраховується ОС в момент вибору процесу для виконання. Він змінюється в залежності від наступних факторів: відносний пріоритет процесу, час очікування запуску, поточний стан процесу та ін.;

– поточний каталог. Кожний процес має свій поточний та робочий каталоги;

– час виконання. Для процесу визначений користувацький, системний і реальний час виконання процесу. Користувацький час – це час, витрачений ЦП на виконання коду програми. Системний час – це час, витрачений ЦП на обробку системних викликів, операцій вводу/виводу, пересилки даних. Реальний час – це час, що пройшов від моменту запуску програми;

– змінні оточення;

– розмір програми – об'єм пам'яті, яку займає процес. Кожний процес характеризується значеннями віртуального розміру і розміру резидентної частини;

Перераховані параметри є спільними для різних операційних систем. Проте цей перелік може бути доповнений деякими параметрами, що є важливими для процесу у конкретній ОС.

Для вирішення задачі прогнозування стану процесу представимо його набором характеристик (сигнатурою).

Визначення 1. Сигнатурою процесу назвемо сукупність його

характеристик, яка однозначно ідентифікує поведінку процесу у комп'ютерній системі.

До характеристик процесу, що формують сигнатуру, віднесемо ідентифікатор процесу, ідентифікатор батьківського процесу, ідентифікатор користувача якому належить процес, пріоритет процесу, квоти процесу (кількість пам'яті і процесорний час доступні процесу), дескриптори відкритих процесом файлів [5,6].

Подамо сигнатуру процесу у наступному вигляді (рис.2.1):

1) ідентифікатор процесу (x_1 біт);

2) ідентифікатор батьківського процесу (x_2-x_1 біт);

3) ідентифікатор користувача (x_3-x_2 біт);

4) пріоритет процесу (x_4-x_3 біт);

5) кількість дескрипторів файлів, що використовуються процесом (x_5-x_4 біт);

6) обсяг віртуальної пам'яті, що використовує процес (x_6-x_5 біт);

7) час виконання процесу (x_7-x_6 біт);

8) додаткові параметри процесу (x_n-x_7 біт).

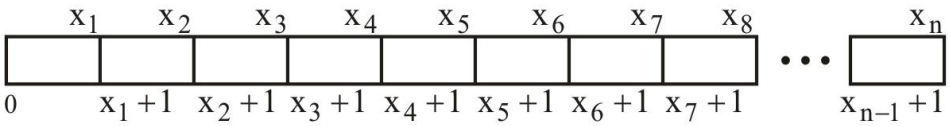


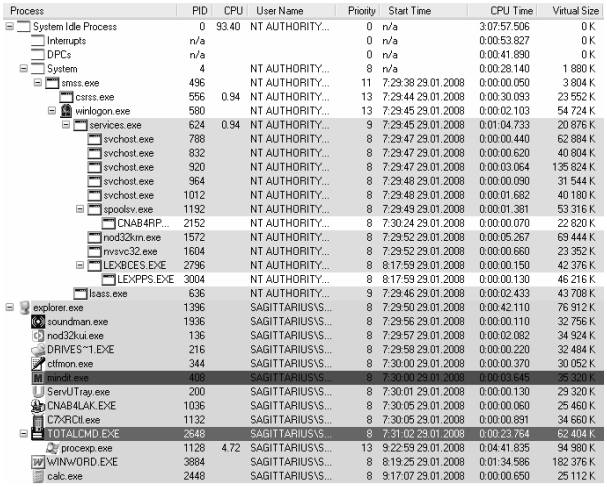
Рисунок 2.1 - Представлення сигнатури у машинному форматі

Довжина сигнатури буде залежати від кількості додаткових параметрів, які будуть включені в неї, та від архітектурних особливостей певної ОС.

Для відображення процесів та відповідних їм сигнатур подамо їх у наступній формі: (I,S,T), де I – ім'я процесу, S – побудована сигнатура для даного процесу, T – час побудови або останньої модифікації сигнатури для даного процесу.

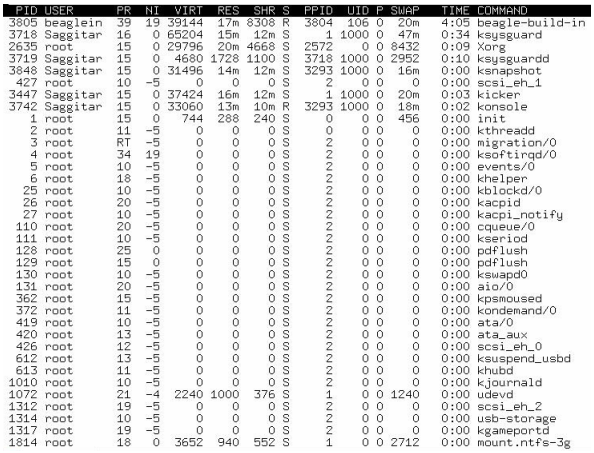
Розглянемо на прикладі реальні процеси в ОС сімейства Windows (рис.2.2) та Linux (рис.2.3).

Включимо до складу сигнатури наступні параметри [8,9], якими володіє процес у даній ОС (табл.2.1):



Process	PID	CPU	User Name	Priority	Start Time	CPU Time	Virtual Size
System Idle Process	0	93.40	NT AUTHORITY...	0	n/a	3:07:57.506	0 K
Interrupts	n/a			0	n/a	0:00:53.827	0 K
DPCs	n/a			0	n/a	0:00:41.890	0 K
System	4		NT AUTHORITY...	8	n/a	0:00:28.140	1 880 K
smss.exe	496		NT AUTHORITY...	11	7:29:38 29.01.2008	0:00:00.050	3 804 K
csrss.exe	556	0.94	NT AUTHORITY...	13	7:29:44 29.01.2008	0:00:30.093	23 552 K
winlogon.exe	580		NT AUTHORITY...	13	7:29:45 29.01.2008	0:00:02.103	54 724 K
services.exe	624	0.94	NT AUTHORITY...	9	7:29:45 29.01.2008	0:01:04.733	20 876 K
svchost.exe	788		NT AUTHORITY...	8	7:29:47 29.01.2008	0:00:00.440	62 884 K
svchost.exe	832		NT AUTHORITY...	8	7:29:47 29.01.2008	0:00:00.620	40 804 K
svchost.exe	920		NT AUTHORITY...	8	7:29:47 29.01.2008	0:00:03.064	135 824 K
svchost.exe	964		NT AUTHORITY...	8	7:29:48 29.01.2008	0:00:00.090	31 544 K
svchost.exe	1012		NT AUTHORITY...	8	7:29:48 29.01.2008	0:00:01.682	40 180 K
spoolsv.exe	1192		NT AUTHORITY...	8	7:29:49 29.01.2008	0:00:01.381	53 316 K
CNAB4RP...	2152		NT AUTHORITY...	8	7:30:24 29.01.2008	0:00:00.070	22 820 K
nod32krn.exe	1572		NT AUTHORITY...	8	7:29:52 29.01.2008	0:00:05.267	69 444 K
nvsvc32.exe	1604		NT AUTHORITY...	8	7:29:52 29.01.2008	0:00:00.660	23 352 K
LEXBCES.EXE	2796		NT AUTHORITY...	8	8:17:59 29.01.2008	0:00:00.150	42 376 K
LEXPPS.EXE	3004		NT AUTHORITY...	8	8:17:59 29.01.2008	0:00:00.130	46 216 K
lsass.exe	636		NT AUTHORITY...	9	7:29:46 29.01.2008	0:00:02.433	43 708 K
explorer.exe	1396		SAGITTARIUS\S...	8	7:29:50 29.01.2008	0:00:42.110	76 912 K
soundman.exe	1936		SAGITTARIUS\S...	8	7:29:56 29.01.2008	0:00:00.110	32 756 K
nod32kui.exe	136		SAGITTARIUS\S...	8	7:29:57 29.01.2008	0:00:02.082	34 924 K
DRIVES~1.EXE	216		SAGITTARIUS\S...	8	7:29:58 29.01.2008	0:00:00.220	32 484 K
ctfmon.exe	344		SAGITTARIUS\S...	8	7:30:00 29.01.2008	0:00:00.370	30 052 K
mindit.exe	408		SAGITTARIUS\S...	8	7:30:00 29.01.2008	0:00:03.645	35 320 K
ServUTray.exe	200		SAGITTARIUS\S...	8	7:30:01 29.01.2008	0:00:00.130	29 320 K
CNAB4LAK.EXE	1036		SAGITTARIUS\S...	8	7:30:05 29.01.2008	0:00:00.060	25 460 K
C7XRDtl.exe	1132		SAGITTARIUS\S...	8	7:30:05 29.01.2008	0:00:00.891	34 660 K
TOTALCMD.EXE	2648		SAGITTARIUS\S...	8	7:31:02 29.01.2008	0:00:23.764	62 404 K
procexp.exe	1128	4.72	SAGITTARIUS\S...	13	9:22:59 29.01.2008	0:04:41.835	94 980 K
WINWORD.EXE	3884		SAGITTARIUS\S...	8	8:19:25 29.01.2008	0:01:34.586	182 376 K
calc.exe	2448		SAGITTARIUS\S...	8	9:17:07 29.01.2008	0:00:00.650	25 112 K

Рисунок 2.2 - Активні процеси в ОС сімейства Windows



PID	USER	PR	NI	VIRT	RES	SHR	S	PPID	UID	P	SWAP	TIME	COMMAND
3805	beaglein	39	19	39144	17m	8308	R	3804	106	0	20m	4:05	beagle-build-in
3718	Saggitar	16	0	65204	15m	12m	S	1	1000	0	47m	0:34	ksysguard
2635	root	15	0	29796	20m	4668	S	2572	0	0	8432	0:09	Xorg
3719	Saggitar	15	0	4680	1728	1100	S	3718	1000	0	2952	0:10	ksysguardd
3848	Saggitar	15	0	31496	14m	12m	S	3293	1000	0	16m	0:00	ksnapshot
427	root	10	-5	0	0	0	S	2	0	0	0	0:00	scsi_eh_1
3447	Saggitar	15	0	37424	16m	12m	S	1	1000	0	20m	0:03	kicker
3742	Saggitar	15	0	33060	13m	10m	R	3293	1000	0	18m	0:02	konsole
1	root	15	0	744	288	240	S	0	0	0	456	0:00	init
2	root	11	-5	0	0	0	S	0	0	0	0	0:00	kthreadd
3	root	RT	-5	0	0	0	S	2	0	0	0	0:00	migration/0
4	root	34	19	0	0	0	S	2	0	0	0	0:00	ksoftirqd/0
5	root	10	-5	0	0	0	S	2	0	0	0	0:00	events/0
6	root	18	-5	0	0	0	S	2	0	0	0	0:00	khelper
25	root	10	-5	0	0	0	S	2	0	0	0	0:00	kblockd/0
26	root	20	-5	0	0	0	S	2	0	0	0	0:00	kacpid
27	root	10	-5	0	0	0	S	2	0	0	0	0:00	kacpi_notify
110	root	20	-5	0	0	0	S	2	0	0	0	0:00	cqueue/0
111	root	10	-5	0	0	0	S	2	0	0	0	0:00	kseriod
128	root	25	0	0	0	0	S	2	0	0	0	0:00	pdflush
129	root	15	0	0	0	0	S	2	0	0	0	0:00	pdflush
130	root	10	-5	0	0	0	S	2	0	0	0	0:00	kswapd0
131	root	20	-5	0	0	0	S	2	0	0	0	0:00	aio/0
362	root	15	-5	0	0	0	S	2	0	0	0	0:00	kpsmoused
372	root	11	-5	0	0	0	S	2	0	0	0	0:00	kondemand/0
419	root	10	-5	0	0	0	S	2	0	0	0	0:00	ata/0
420	root	13	-5	0	0	0	S	2	0	0	0	0:00	ata_aux
426	root	12	-5	0	0	0	S	2	0	0	0	0:00	scsi_eh_0
612	root	13	-5	0	0	0	S	2	0	0	0	0:00	ksuspend_usbd
613	root	11	-5	0	0	0	S	2	0	0	0	0:00	khubd
1010	root	10	-5	0	0	0	S	2	0	0	0	0:00	kjournald
1072	root	21	-4	2240	1000	376	S	1	0	0	1240	0:00	udevd
1312	root	19	-5	0	0	0	S	2	0	0	0	0:00	scsi_eh_2
1314	root	10	-5	0	0	0	S	2	0	0	0	0:00	usb-storage
1317	root	19	-5	0	0	0	S	2	0	0	0	0:00	kgameportd
1814	root	18	0	3652	940	552	S	1	0	0	2712	0:00	mount.ntfs-3g

Рисунок 2.3 - Активні процеси в ОС сімейства Linux

Таблиця 2.1 - Параметри процесу, що входять до складу сигнатури

№ п/п	Параметр	ОС Windows		ОС Linux	
		Використання	Кількість бітів	Використання	Кількість бітів
1	ідентифікатор процесу	+	14	+	14
2	ідентифікатор батьківського процесу	+	14	+	14
3	ідентифікатор користувача	+	4	+	10
4	пріоритет процесу	+	4	+	8
5	кількість дескрипторів файлів, що використовуються процесом	+	8	+	8
6	обсяг віртуальної пам'яті, що використовує процес	+	30	+	32
7	час виконання процесу	+	30	+	32
8	додатковий пріоритет процесу	+	4	+	8
9	стан процесу	-	-	+	2

Результати побудови сигнатур для двох процесів в ОС Windows та для двох процесів в ОС Linux представимо у таблиці 2.2.

Таблиця 2.2 - Процеси та відповідні їм побудовані сигнатури

Ім'я процесу	ОС	Система числення	Сигнатура процесу	Час створення
nod32kui.exe (антивірусна система NOD32)	Windows XP	двійкова	00000010001000 00010101110100 0011 1000 01000110 00000000000000001000100001101100 00000000000000000000100000100010 1001	14:15:07
		шістнадцяткова	022057438460000886C000008229	
TOTALCMD.EXE (файловий менеджер)	Windows XP	двійкова	00101001011000 00010101110100 0011 1000 10101000 00000000000000001111001111000100 00000000000000000101110011010100 1000	14:15:07
		шістнадцяткова	296057438A80000F3C400005CD48	
ksysguard (менеджер процесів)	Linux SUSE 10.3	двійкова	00111010000110 00000000000001 1111101000 00010000 00000001 00000000000000001111111010110100	15:06:45
		шістнадцяткова	3A18001FA040040003FAD00001735001	
Xorg (графічна система)	Linux SUSE 10.3	двійкова	00101001001011 00101000001100 0000000000 00001111 00000101 00000000000000000111010001100100	15:06:45
		шістнадцяткова	292CA0C0003C180001D1900000208805	

2.2 Модель життєвого циклу процесів.

Процес – це система дій, що реалізує певну функцію в обчислювальній системі й оформлена так, що керуюча програма обчислювальної системи може перерозподіляти ресурси цієї системи з метою забезпечення багатозадачності. Позначимо множину виконуваних процесів, як $A = \{a_i\}_{i=1}^{y}$, де у – кількість процесів.

Ресурс обчислювальної системи – засіб обчислювальної системи, що може бути виділений процесу обробки даних на певний інтервал часу. Позначимо множину наявних ресурсів ОС, як $RE = \{re_j\}_{j=1}^{x}$, де х – кількість видів ресурсів.

До ресурсів ОС віднесемо наявну пам'ять, процесори, пристрої вводу/виводу, а також дані, необхідні для роботи процесів (файли в пам'яті та на зовнішніх носіях, результати обчислень інших процесів).

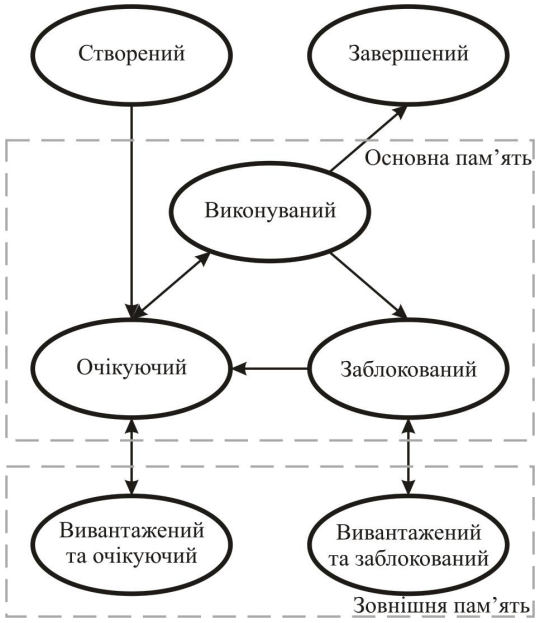


Рисунок 2.4 - Діаграма станів процесу

Кожен процес від моменту створення до моменту завершення проходить через ряд станів (рис. 2.4). Проте така діаграма станів не відображає стан взаємоблокування процесів.

Під сигнатурою процесу будемо розуміти сукупність його характеристик, яка однозначно ідентифікує стан процесу в ОС в певний момент часу t:

$$a_i(t) \to (a_{i_1}^t, a_{i_2}^t, ..., a_{i_z}^t), \tag{2.1}$$

де $a_i(t) \in A$ – поточний процес, $a_{i_1}^t, \ldots, a_{i_z}^t$ – характеристики процесу в поточний момент часу (параметри та ресурси, які використовує процес в даний момент).

До характеристик процесу, що формують сигнатуру, віднесемо наступні: ідентифікатор процесу, ідентифікатор батьківського процесу, ідентифікатор користувача, якому належить процес, пріоритет процесу, квоти процесу (кількість пам'яті і процесорний час доступні процесу), дескриптори відкритих процесом файлів.

Оскільки до складу сигнатури процесу входять унікальні характеристики, то в один і той самий момент часу в системі не існує двох абсолютно однакових сигнатур.

Отже, життєвий цикл процесу можна подати у вигляді послідовності станів, через які проходить цей процес. Перехід із стану в стан відбувається через зміну певних параметрів, якими характеризується процес. Зміна параметрів процесу відбувається з ряду причин: дії ОС, дії інших процесів, виконання власного програмного коду. Стан кожного окремого процесу буде впливати на стан КС в цілому.

Позначимо стан процесу через w, причину зміни параметрів через r, а перехід із стану в стан через $w_i \xrightarrow{\textit{зміна}\ \text{параметрів з причини}\ r_j} w_{i+1}$. Тоді життєвий цикл процесу буде мати вигляд (2):

$$a_i : w_0 \xrightarrow{r_j} w_1 \xrightarrow{r_j} \ldots \xrightarrow{r_j} w_{k-1} \xrightarrow{r_j} w_k, \tag{2.2}$$

де $w_0 \in W$ – початковий стан процесу (стан "створений"), $w_k \in W$ – кінцевий стан процесу (стан "завершений"), W – множина програмних станів процесу (рис.2.4), $r_j \in R$ - множина можливих причин зміни параметрів процесу (j=1,2,3…).

Враховуючи визначення сигнатури та (2.1) і (2.2), життєвий цикл кожного процесу можна подати у вигляді:

$$a_i : (a_{i_1}^{t_0}, a_{i_2}^{t_0}, ..., a_{i_z}^{t_0}) \xrightarrow{\mathrm{r_j}} (a_{i_1}^{t_1}, a_{i_2}^{t_1}, ..., a_{i_z}^{t_1}) \xrightarrow{\mathrm{r_j}} \xrightarrow{\mathrm{r_j}} (a_{i_1}^{t_{k-1}}, a_{i_2}^{t_{k-1}}, ..., a_{i_z}^{t_{k-1}}) \xrightarrow{\mathrm{r_j}} (a_{i_1}^{t_k}, a_{i_2}^{t_k}, ..., a_{i_z}^{t_k}), \quad (3)$$

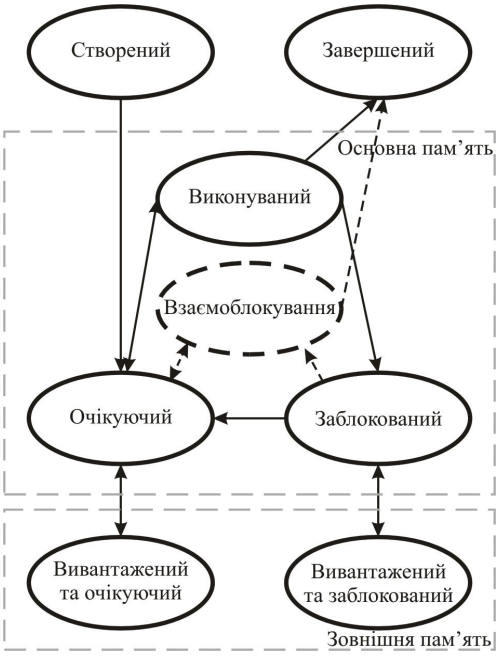


Рисунок 2.5 - Діаграма станів процесу, що включає стан взаємоблокування

У стан взаємоблокування можуть потрапляти процеси, що взаємодіють між собою у багатозадачних ОС в певний момент часу. До потрапляння у стан взаємоблокування процеси протягом свого життєвого циклу знаходяться в інших станах, а саме стан "створений", стан "очікуючий", стан "виконуваний", стан "заблокований", стан "завершений" (рис.2.5). У стан взаємоблокування процеси потрапляють, як правило, із

стану "заблокований" або стану "очікуючий". Отже, у даний момент часу серед множини процесів можна виділити підмножину процесів, які можуть в наступний момент часу потрапити до стану взаємоблокування. Перед входженням у стан взаємоблокування процес буде знаходитись у певному "граничному" стані, після якого ймовірність переходу у стан взаємоблокування буде високою. Взаємоблокування процесів призводить до часткової або повної втрати функційної здатності ОС. Тому вважатимемо стан взаємоблокування процесів неробочим станом ОС, а інші стани процесів – робочим станом ОС.

Віднесемо до робочого стану такі стани процесу: стан "створений", стан "виконуваний", стан "завершений". До "граничного" стану віднесемо такі стани процесу: стан "заблокований", стан "очікуючий".

Представимо шлях, яким процес потрапляє у стан взаємоблокування, у вигляді наступної схеми (рис.2.6).

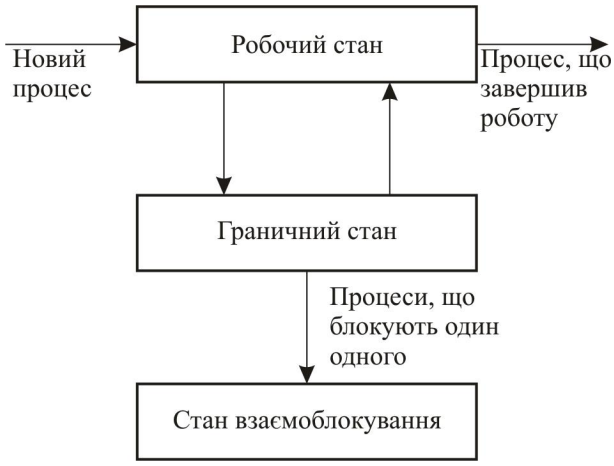


Рисунок 2.6 - Схема переходу процесів у стан взаємоблокування

Як видно із схеми, виникнення взаємоблокування процесів можливе лише для частини процесів, що знаходяться у граничному стані. При переході процесів до граничного стану відбувається зміна їхніх параметрів.

Розглянемо життєвий цикл процесу. В момент створення (стан "створений") процес знаходиться у робочому стані., йому надана частина

ресурсів системи. Позначимо цей стан процесу, як $s_{роб}(t)$. В певний момент часу процесу для подальшого виконання необхідний додатковий ресурс системи, який на даний час є недоступний. Відбувається перехід процесу до стану "заблокований", тобто процес потрапляє у граничний стан. Позначимо цей стан процесу, як $s_{гран}.(t)$, а перехід до даного стану, як $s_{роб}(t) \xrightarrow{\text{потреба ресурсу } re_i} s_{гран}(t)$. При задоволенні потреб процесу у ресурсах він повертається у робочий стан, тобто здійснює перехід $s_{гран}(t) \xrightarrow{\text{надання ресурсу } re_i} s_{роб}(t)$. Коли необхідний ресурс отримати неможливо по причині його використання іншим процесом, то процес залишається у граничному стані. Якщо ж другий процес в свою чергу очікує ресурс, що зайнятий першим процесом, то вони обидва потрапляють у стан взаємоблокування. Позначимо цей стан процесу, як $s_{взаємоблокування}.(t)$, а перехід до даного стану, як $s_{гран}(t) \xrightarrow[\text{утримування ресурсу } re_j]{\text{очікування ресурсу } re_i} s_{взаємоблокування}(t)$.

Таким чином, враховуючи (3), життєвий цикл процесу буде мати один з наступних виглядів:

процес створений, йому надано всі необхідні для завершення роботи ресурси, він виконується і завершується (процес весь час знаходиться в робочому стані sроб(t)), тобто:

$$a_i : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k, \qquad (2.4)$$

де $s_0 \in s_{роб}$, $s_1 \in s_{роб}$, $s_k \in s_{роб}$.

процес створений, йому надано частину ресурсів, необхідних для завершення роботи, він потребує додаткових ресурсів, через деякий час отримує їх, виконується і завершується (процес спочатку знаходиться в робочому стані $s_{роб}(t)$, потім переходить $s_{роб}(t) \xrightarrow{\text{потреба ресурсу } re_i} s_{гран}(t)$

в граничний стан sгран(t), потім $s_{\text{гран}}(t) \xrightarrow{\text{надання ресурсу } re_i} s_{\text{роб}}(t)$ знову в робочий стан $s_{\text{роб}}(t)$), тобто:

$$a_i : s_0 \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_l \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_k, \tag{2.5}$$

де $s_0 \in s_{роб}$, $s_l \in s_{гран}$, $s_k \in s_{роб}$.

процес створений, йому надано частину ресурсів, необхідних для завершення роботи, він потребує додаткових ресурсів, які утримує інший процес, який в свою чергу потребує ресурсів першого процесу (процес спочатку знаходиться в робочому стані sроб(t), потім переходить $s_{\text{роб}}(t) \xrightarrow{\text{потреба ресурсу } re_i} s_{\text{гран}}(t)$ в граничний стан sгран(t), із якого відбувається перехід $s_{\text{гран}}(t) \xrightarrow[\text{утримування ресурсу } re_j]{\text{очікування ресурсу } re_i} s_{\text{взаємоблокування}}(t)$ до стану взаємоблокування).

$$\begin{cases} a_i : s_0 \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_l \xrightarrow{r_j} s_x \\ a_m : s_0 \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_n \xrightarrow{r_j} s_x \end{cases}, \tag{2.6}$$

де $s_0 \in s_{роб}$, $s_l \in s_{гран}$, $s_n \in s_{гран}$, $s_x \in s_{взаємоблокування}$.

Із розглянутих вище можливих варіантів поведінки процесів критичним для ОС є останній, при якому процеси потрапляють у стан взаємоблокування

2.3 Модель процесу прогнозування стану процесу в персональному комп'ютері

Подамо процес прогнозування стану процесів в ПК наступною моделлю (2.7):

$$M = \langle A, S, D, P, R \rangle \qquad (2.7)$$

де A - множина сигнатур процесів, що виконуються на ПК у даний момент;

S – впорядкована послідовність характеристик комп'ютерної системи (загальний обсяг оперативної пам'яті, зовнішньої пам'яті, обсяг вільної в даний момент пам'яті, кількість периферійних пристроїв, кількість активних процесів користувача та ядра системи);

D - підмножина сигнатур процесів, що знаходяться у стані, наближеному до стану взаємоблокування;

P - множина правил, на основі яких проводиться розбиття множини сигнатур працюючих процесів на дві підмножини: підмножину сигнатур процесів, що знаходяться у стані, наближеному до стану взаємоблокування, та підмножину сигнатур процесів, які не досягли цього стану;

R - вектор ймовірностей переходу у стан блокування процесів із підмножини D.

Для забезпечення процесу прогнозування стану процесів в КС включимо до моделі наступні структурні частини (рис.2.7):

- підсистема виявлення зміни стану процесу – призначена для контролю за поточними параметрами процесів, що вже присутні в КС, та запуском нових процесів;

- підсистема визначення характеристик КС – призначена для контролю за зміною основних характеристик КС, що суттєво впливають на наближення процесів до стану взаємоблокування;

- підсистема побудови сигнатури процесу – призначена для побудови сигнатури процесу, що виконує перехід у стан готовності і ще не

має сигнатури, та модифікації сигнатури процесів, що змінили значення своїх параметрів та мають відповідні їм сигнатури;

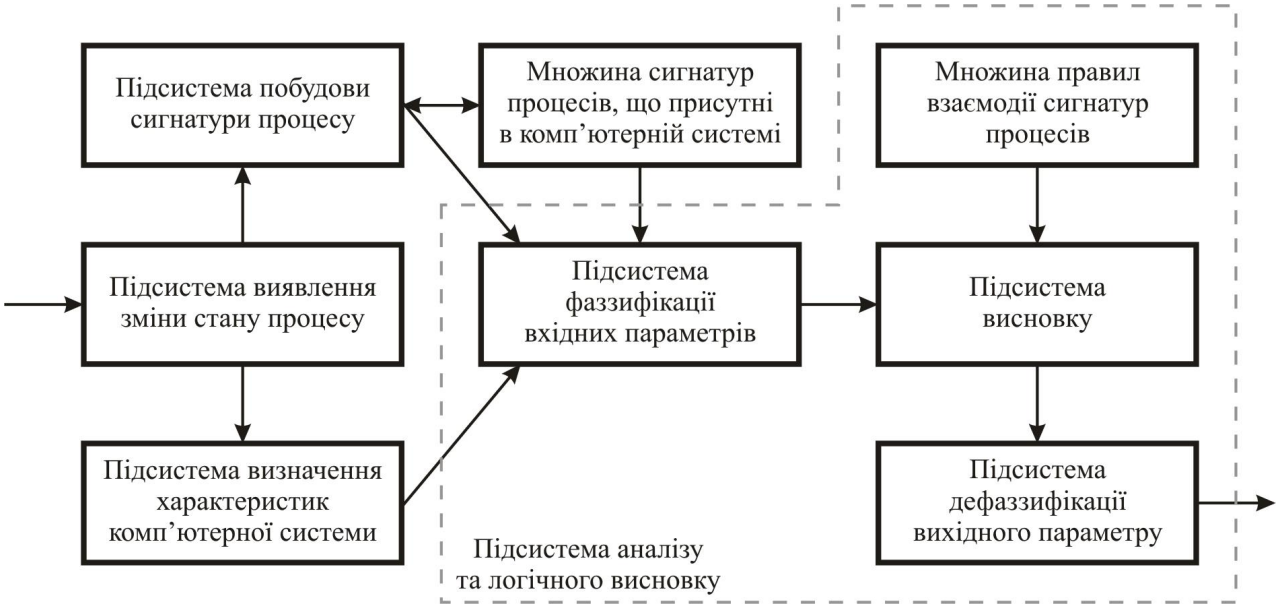


Рисунок 2.7 - Модель прогнозування стану процесів в комп'ютерній системі

- множина сигнатур процесів, що присутні у КС – призначена для зберігання сигнатур процесів, які в даний момент виконуються на КС;

- підсистема аналізу та логічного висновку – за допомогою множини правил взаємодії сигнатур процесів виділяє із множини сигнатур всіх процесів підмножину сигнатур процесів, що наближаються до стану взаємоблокування, і робить висновок про можливість продовження роботи цих процесів.

Для здійснення аналізу та логічного висновку використаємо нечітку експертну систему. Згідно з [5,6] включимо до складу підсистеми аналізу та логічного висновку наступні компоненти:

- підсистема фаззифікації вхідних параметрів – визначає ступені впевненості в тому, що вихідні лінгвістичні змінні отримають конкретні значення;

- множина правил взаємодії сигнатур процесів – містить правила, що визначають поведінку процесів за їх сигнатурами;

- підсистема висновку – на основі множини правил взаємодії сигнатур процесів та набору вхідних лінгвістичних змінних проводиться оцінка істинності для кожного правила та формується єдина нечітка множина;

- підсистема дефаззифікації вихідного параметру – перетворює нечіткий набір значень вихідної лінгвістичної змінної до точного значення.

2.4 Підсистема аналізу та логічного висновку

Позначимо через U множину всіх можливих сигнатур процесів (універсум). Виділимо множину сигнатур процесів, що є активними в даний момент в КС і позначимо її через A ($A \subset U$). Процеси, що є активними в даний момент, можна розділити на 2 підмножини: процеси, запущені користувачем, та процеси, запущені ядром системи. Отже, множину А можна розбити на дві підмножини: K та S ($K \subset U, S \subset U, K \bigcup S = A$). Всі перераховані множини належать до чітких. Їх характеристична функція приймає 1, якщо елемент належить множині, і 0 в протилежному випадку.

При наближенні до стану взаємоблокування процесів в КС із множини A можна виділити підмножину A^* сигнатур процесів, які найімовірніше приведуть до цього стану. Відповідно множину A^* також можна розділити на 2 підмножини: K^* - множина процесів, запущених користувачем, які ймовірно призведуть до стану блокування, та S^* - множина процесів, запущених ядром системи, які ймовірно призведуть до блокування ($A^* \subset U, K^* \subset U, S^* \subset U, K^* \bigcup S^* = A^*$). Множини A^*, K^*, S^* є нечіткими, оскільки сигнатури процесів входять у них з певними ймовірностями. Отже, для реалізації висновку про наближення до стану взаємоблокування процесів необхідно скористатись апаратом нечіткого висновку.

Оскільки взаємоблокування настає по причині конкуренції процесів за системні ресурси, необхідно включити їх кількісні характеристики до бази знань експертної системи. Врахуємо наступні системні ресурси:

- кількість процесів користувача, присутніх в КС;
- кількість процесів ядра системи, присутніх в КС;
- обсяг ОП в КС (загальний та вільної в даний момент);
- обсяг ЗП в КС (загальний та вільної в даний момент);
- кількість пристроїв вводу-виводу інформації (загальна та вільних у даний момент);
- загальна кількість файлів у КС.

Подамо процес прогнозування стану процесів у КС за результатами експертних оцінок у вигляді множини $S = \{s_{1,}, s_2, ..., s_i, ..., s_n\}$, де s_i- ймовірність настання блокування даного процесу.

Для множини S побудуємо множину функцій належності $M = \{\mu_{s_{1,}}, \mu_{s_2}, ..., \mu_{s_i}, ..., \mu_{s_n}\}$, яка відображатиме ступінь впевненості експерта у істинності твердження стосовно ймовірності блокування даного процесу.

В якості функції належності для μ_{s_i} оберемо трапецевидну функцію із класу кусочно-лінійних, оскільки вона використовується для задання властивостей множин, які характеризують невизначеність типу "приблизно рівно", "середнє значення", "схожий до об’єкту", "подібний об’єкту" [7]. Трапецевидну функція належності задамо як в [6].

Виберемо для показників наступну терм-множину: {мало, середнє значення, багато}. Відповідно до такого представлення їх функції належності будуть мати наступний вигляд (рис.2.8):

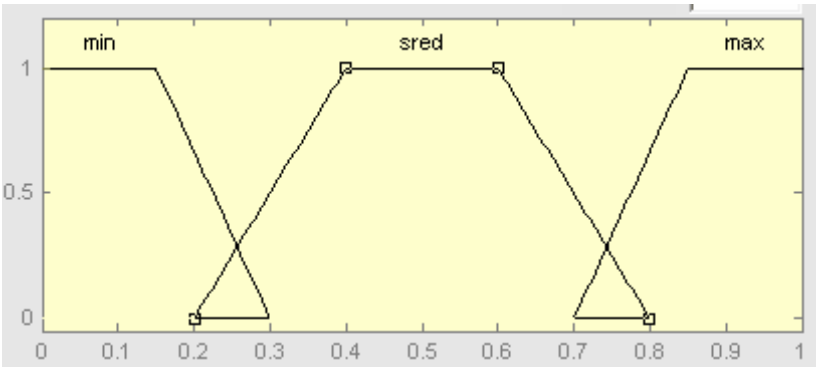


Рисунок 2.8 - Функції належності вхідних параметрів.

Подамо ступені впевненості експерта в ймовірності блокування процесу наступним чином:

- дуже низька ймовірність – 0,1
- низька ймовірність – 0,3
- середня ймовірність – 0,5
- висока ймовірність – 0,7
- дуже висока ймовірність – 0,9

Така система оцінок при використанні алгоритму Мамдані дозволить здійснити визначення ймовірності блокування процесів в КС шляхом здійснення нечіткого логічного висновку [8].

Задамо вхідні змінні у вигляді лінгвістичних та визначимо їх функції належності.

У якості вхідних змінних використаємо показники системних ресурсів та сигнатур активних процесів.

Задамо межі значень для кожного виду ресурсів. Для ресурсів 1 та 2 значення будуть знаходитись в діапазоні від 0 до K (K – максимальна кількість процесів в конкретній КС); для ресурсів 3-4 – в межах від 0 до n (n – загальний обсяг ОП та ЗП в конкретній КС); для ресурсу 5 – в межах від 0 до m (m – загальна кількість пристроїв вводу-виводу в конкретній КС); для ресурсу 6 – в межах від 0 до k.

Оскільки всі вхідні характеристики знаходяться у різних числових межах, що є незручним для опрацювання результатів, то проведемо

нормування даних величин. Зведемо усі показники до меж [0;1]. Для цього необхідно виконати ділення показника на максимально допустиме значення цього показника в конкретній КС. Наприклад, загальний об'єм ОП складає 256 Мб, а вільної в даний момент пам'яті – 60 Мб. Тоді показник вільної в даний момент пам'яті складе 60 / 256 = 0,234.

Проте такий підхід до нормування показників не буде враховувати ступінь впливу даного показника на результат системи прогнозування стану процесів, оскільки для показників із однаковим нормованим значенням, але з різними поточними та максимально допустимими значеннями, буде різний вплив на КС. Наприклад, при нормованому значенні показника вільної пам'яті 0,1 для КС із загальним об'ємом ОП 2Гб та КС із загальним об'ємом ОП 128 Мб кількісні показники значно відрізняються і є більш критичними для КС із меншим об'ємом ОП. Тому необхідно при нормуванні враховувати максимально допустиме значення показника. Для цього нормування показників проведемо за наступною формулою:

$$P_n = 1 - \frac{P_d + P_m}{P_d \cdot P_m}, \quad Pd \neq 0, \tag{2.8}$$

де P_n – черговий нормований показник;

P_d – поточне значення показника в конкретній КС;

P_m – максимальне значення показника в конкретній КС.

Такий підхід до нормування показників дозволяє врахувати ступінь впливу даного показника на результат системи прогнозування стану процесів.

2.5 Висновки

Розроблено сигнатурну модель процесу, визначено перелік параметрів процесу, які включені в структуру сигнатури процесу.

Запропоновано модель прогнозування стану процесів в КС, яка дає можливість виділити підмножину сигнатур процесів, що наближаються до

стану взаємоблокування, та зробити висновок про можливість подальшої роботи цих процесів.

Проведено дослідження адекватності моделі з використанням імітаційного моделювання

На відміну від відомих підходів запропонована модель дозволяє уникнути наявності циклів активного очікування, блокування роботи ОС та необхідності використання спеціалізованої команди процесора, що дає можливість для розробки методу та засобів, які дозволять попереджувати взаємоблокування процесів в комп'ютерній системі

3 МЕТОД ТА АЛГОРИТМИ ПРОГНОЗУВАННЯ СТАНУ ПРОЦЕСІВ В ПЕРСОНАЛЬНОМУ КОМП'ЮТЕРІ

3.1 Метод прогнозування стану процесу

На основі моделі прогнозування стану процесів метод прогнозування стану процесів в ПК включає наступні етапи:

1) виявлення змін в системі;

2) побудова сигнатури процесів;

3) визначення основних характеристик ПК та ОС;

4) приведення до нормованого вигляду вхідних параметрів та їх передача на підсистему аналізу та логічного висновку;

5) здійснення висновку про настання взаємоблокування процесів

На першому етапі виявляємо зміни, що відбуваються в системі, а саме виявляємо запити на створення нового процесу чи зміну параметрів вже існуючого процесу. Ця робота покладена на підсистему виявлення зміни стану процесу.

На другому етапі здійснюємо побудову (перебудова для вже існуючих процесів) сигнатури для виявленого процесу. Ці дії покладені на підсистему побудови сигнатури.

На третьому етапі визначаємо поточні характеристики ПК та ОС. Ці дії проводяться підсистемою визначення характеристик ПК та ОС.

Оскільки взаємоблокування настає по причині конкуренції процесів за системні ресурси, то включаємо їх кількісні характеристики до бази знань експертної системи:

- кількість процесів користувача, присутніх на ПК в певній ОС;
- кількість процесів ядра системи, присутніх на ПК в певній ОС;
- обсяг ОП на ПК в певній ОС (загальний та вільної в даний момент);
- обсяг ЗП на ПК в певній ОС (загальний та вільної в даний момент);

- кількість пристроїв вводу-виводу інформації (загальна та вільних у даний момент);
- загальну кількість файлів на ПК в певній ОС.

На четвертому етапі передаємо множину сигнатур процесів та множину характеристик ПК та ОС для проведення аналізу на предмет настання ситуації взаємоблокування в системі. Ці дані подаються на підсистему аналізу та логічного висновку.

На п'ятому етапі відбувається аналіз взаємодії процесів на основі аналізу їхніх сигнатур та характеристик ПК та ОС і здійснюється висновок про настання ситуації взаємоблокування процесів. Для здійснення даних дій використовується нечітка експертна система [6].

Якщо новий процес за висновком системи не призводить до взаємоблокування, то йому дозволяється виконання на ПК і його сигнатура додається до бази сигнатур працюючих процесів. В протилежному випадку система виявляє множину процесів, які наближаються до стану взаємоблокування, та приймає рішення, який із процесів зняти із виконання.

Послідовність наведений вище дій можна подати наступним алгоритмом (рис.3.1):

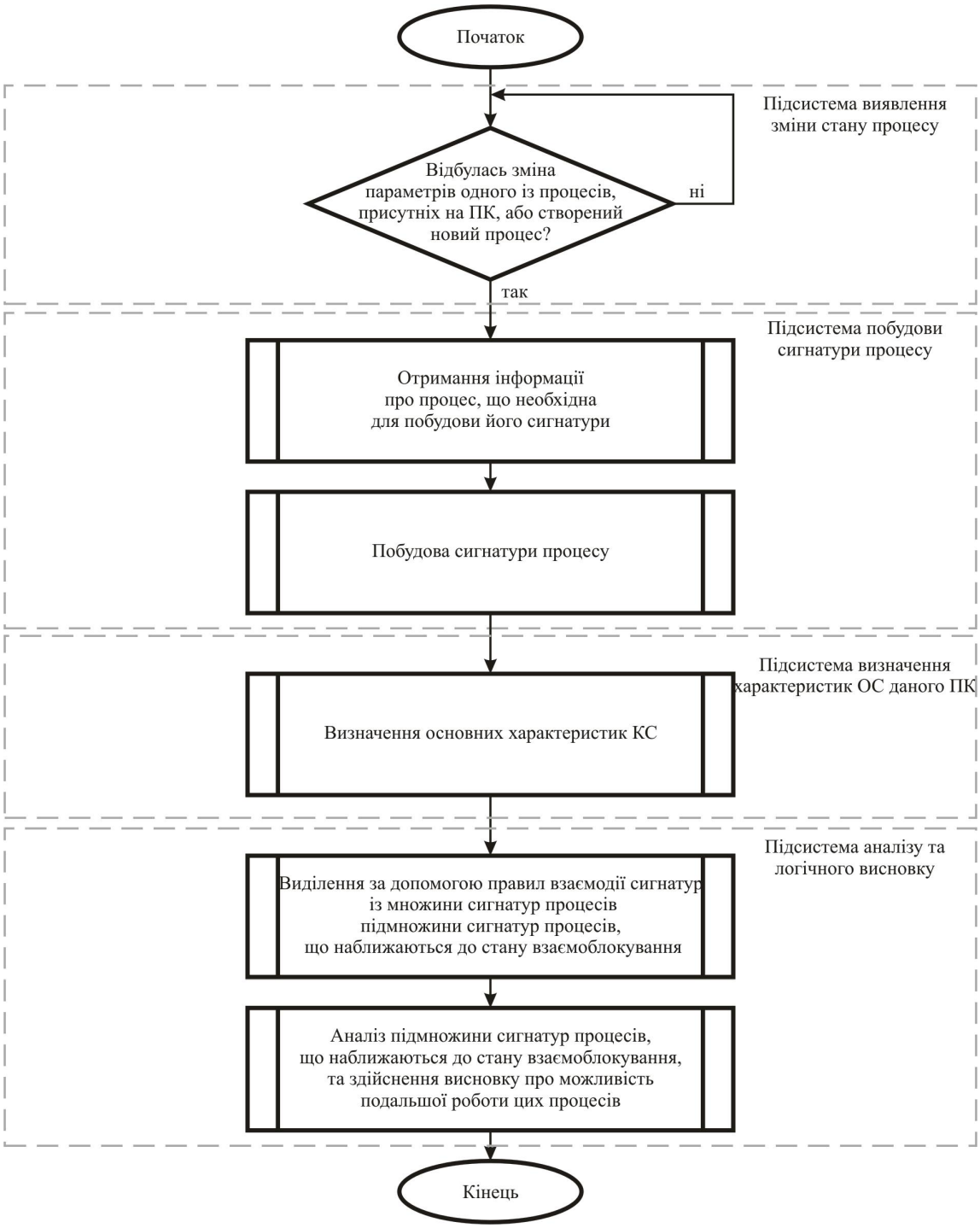


Рисунок 3.1 - Алгоритм визначення стану процесу

3.2 Алгоритми прогнозування стану процесу

Як видно із рис.2.6, прогнозування стану процесу включає два етапи: визначення множини процесів, що можуть потрапити у стан взаємоблокування; виділення із множини потенційних процесів групи процесів, що потраплять у стан взаємоблокування. Таким чином, алгоритм прогнозування стану процесу буде містити дві частини.

Згідно [6] до моделі прогнозування стану процесів входять наступні величини:

A - множина сигнатур процесів, що виконуються на ПК у даний момент;

S– впорядкована послідовність характеристик комп'ютерної системи (загальний обсяг оперативної пам'яті, зовнішньої пам'яті, обсяг вільної в даний момент пам'яті, кількість периферійних пристроїв);

D- підмножина сигнатур процесів, що знаходяться у стані, наближеному до стану взаємоблокування (у граничному стані);

P - множина правил, на основі яких визначається група процесів, що потраплять у стан взаємоблокування;

R- вектор ймовірностей переходу у стан взаємоблокування процесів із підмножини D.

Алгоритм 1. Виявлення потенційних процесів, що можуть потрапити у стан взаємоблокування (виявлення процесів, що знаходяться у граничному стані).

1.1.Якщо $a_k \in A$ потребує ресурс $s_i \in S$, то перехід до 1.2.

1.2.Якщо $a_k \in A$ знаходиться в стані $P_{роб}(t)$, то перехід до 1.3, інакше перехід до 1.4.

1.3.Виконати для $a_k \in A$ перехід із робочого стану до граничного $P_{роб}(t) \xrightarrow{\text{потреба ресурсу } s_i} P_{гран}(t)$ та включити $a_k \in A$ до $D \subset A$. Перехід до 1.4.

1.4.Якщо $a_k \in A$ надано у використання ресурс $s_i \in S$, то перехід до 1.5, інакше перехід до 1.6.

1.5.Виконати для $a_k \in A$ перехід із граничного стану до робочого $P_{гран}(t) \xrightarrow{\text{надання ресурсу } s_i} P_{роб}(t)$ та виключити $a_k \in A$ із $D \subset A$. Перехід до 1.6.

1.6.Якщо перевірено всю множину A, то перехід до 1.7, інакше

перехід до 1.1.

1.7.Кінець алгоритму 1.

Для визначення процесів, що потраплять у стан взаємоблокування, використовується система прогнозування стану процесів, в основі якої лежить система нечіткого логічного висновку (СНЛВ) [7]. На рис.3.2 показана структура СНЛВ.

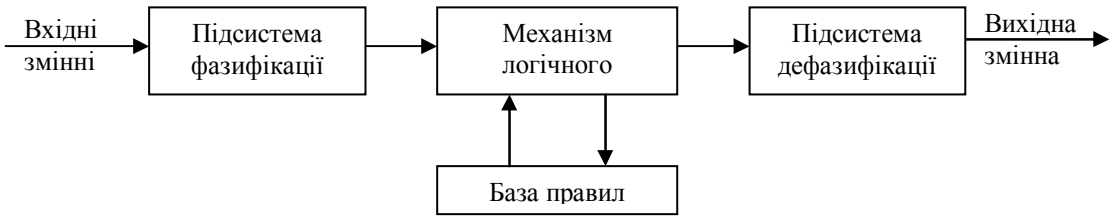


Рисунок 3.2 - Загальна схема СНЛВ

Підсистема фазифікації призначена для визначення ступеня приналежності вхідних значень $x_g \in X$, $X = D \cup S$, $g = \overline{1,h}$ до нечітких множин входу, що являють собою лінгвістичні змінні з відповідної лінгвістичної шкали $T_{x_g} = \left\{ T_{x_g}^1, T_{x_g}^2, ..., T_{x_g}^{m_{x_g}} \right\}$, де m_{x_g} - кількість лінгвістичних змінних у шкалі для g-го входу. Необхідність у фазифікації зумовлена тим, що у СНЛВ використовуються лінгвістичні правила [9].

База правил, що містить лінгвістичні правила, є основою механізму логічного висновку. Механізм логічного висновку здійснює відображення вхідних нечітких множин T_{x_g} за допомогою кожного правила у вихідну T_y з набору вихідних лінгвістичних змінних $T_y = \left\{ T_y^1, T_y^2, ..., T_y^{m_y} \right\}$. Правила із множина правил $P = \left\{ P_j \right\}, j = \overline{1,n}$, що міститься в базі правил, подані у наступному форматі [9,10]:

$$P_j = \text{"}якщо\ x_1\ є\ T_{x_1}\ і\ x_2\ є\ T_{x_2} ... і\ x_h\ є\ T_{x_h}, то\ y_j\ є\ T_y\text{"} \qquad (3.1)$$

Вихідні нечіткі множини y_j кожного правила об'єднуються в одну нечітку множину висновку $\tilde{y}$. Після цього підсистема дефазифікації здійснить відображення нечіткої множини висновку $\tilde{y}$ у чітке число $\overline{y}$, яке буде результатом СНЛВ для заданих вхідних значень x_g.

Алгоритм 2. Виявлення процесів, що потраплять у стан взаємоблокування:

2.1.Проводимо нормування показників за наступною формулою:

2.2.

$$P_n = 1 - \frac{P_d + P_m}{P_d \cdot P_m}, \ Pd \neq 0, \tag{3.2}$$

де P_n – черговий нормований показник;

P_d – поточне значення показника в конкретній системі;

P_m – максимальне значення показника в конкретній системі.

2.3.Для кожного $x_g \in X$, $X = D \cup S$, $g = \overline{1,h}$ визначаємо ступінь належності до нечітких множин входу (ступені істинності $\mu_g^j(x_g)$).

2.4.На основі ступенів істинності передумов $\mu_g^j(x_g)$ для кожного правила $P_j, j = \overline{1,n}$ розраховуємо ступінь його виконання α_j за формулою (3.3).

2.5.

$$\alpha_j = \min\left(\mu_1^j(x_1), \mu_2^j(x_2), ..., \mu_h^j(x_h)\right) \tag{3.3}$$

2.6.На основі ступеню виконання α_j для кожного правила $P_j, j = \overline{1,n}$ розраховуємо результат його виконання.

2.7.На основі результату виконання кожного правила $P_j, j = \overline{1,n}$ визначаємо вихідну нечітку множину з усіченою функцією приналежності

$\ddot{\mu}^j(y)$ за формулою (3.4).

$$\ddot{\mu}^j(y) = \min\left(\alpha_j, \mu^j(y)\right) \tag{3.4}$$

2.8.Вихідні нечіткі множини $\ddot{\mu}^j(y)$ згідно (3.5) агрегуємо в нечітку множину висновку $\tilde{y}$, що має функцію приналежності (3.6).

$$\tilde{y} = \max\left(\mu^j(y)\right), j = \overline{1,r} \tag{3.5}$$

$$\mu_{\tilde{y}} = \max\left(\ddot{\mu}^1(y), \ddot{\mu}^2(y), ..., \ddot{\mu}^r(y)\right) \tag{3.6}$$

2.9.Приводимо до чіткості нечітку множину $\tilde{y}$ за допомогою процедури дефазифікації центроїдним методом (3.7)

$$\overline{y} = \frac{\int_{C_1}^{C_t} x \cdot f_{\tilde{y}}(x)dx}{\int_{C_1}^{C_t} f_{\tilde{y}}(x)dx} \tag{3.7}$$

2.10. Встановлюємо для R_k значення $\overline{y}$.

2.11. Повторюємо кроки 2.1-2.8 k разів.

2.12. Кінець алгоритму 2.

3.2.1 Побудова сигнатури процесу

Представимо процес побудови сигнатури процесу наступним алгоритмом (рис.3.3):

Алгоритм 3.1.

3.1.1. Перевірка, чи опрацьована уся множина процесів. Якщо так, то перехід до 3.1.8, інакше – перехід до 3.1.2.

3.1.2. Перевірка, чи має процес раніше сформовану сигнатуру. Якщо так, то перехід до 3.1.6, інакше перехід до 3.1.3.

3.1.3. Отримання інформації про процес, яка необхідна для формування його сигнатури.

3.1.4. Формування сигнатури процесу. На цьому кроці відбувається перетворення значень необхідних параметрів процесу до заданого вигляду і утворення його сигнатури.

3.1.5. Додавання створеної сигнатури до множини сигнатур процесів. Перехід до 3.1.1.

3.1.6. Перевірка, чи змінились значення складових сигнатури процесу. Якщо так, то перехід до 3.1.7, інакше перехід до 3.1.1.

3.1.7. Модифікація сигнатури процесу згідно поточних значень складових та перехід до 3.1.1.

3.1.8. Збереження поточної множини сигнатур процесів.

3.1.9. Кінець алгоритму

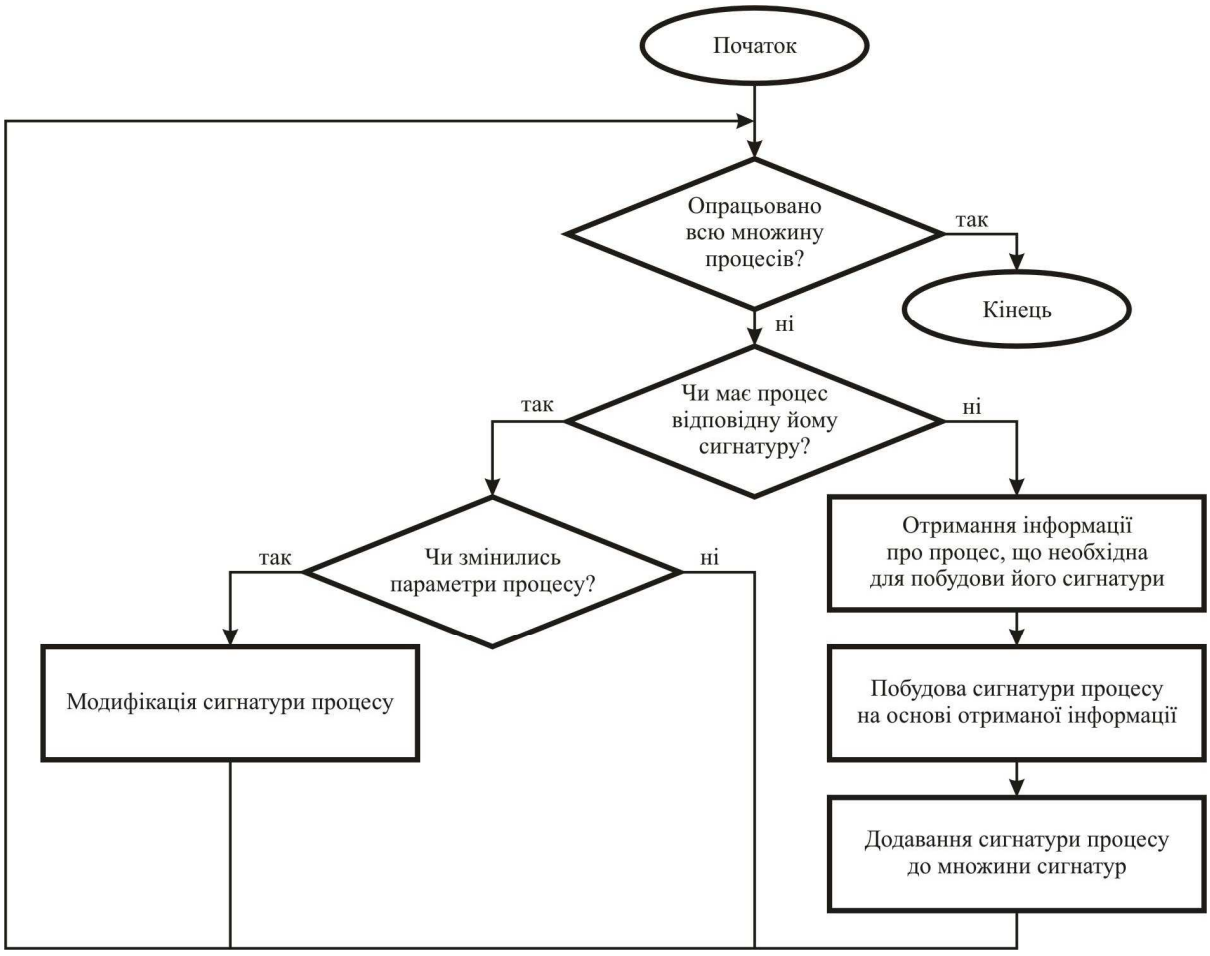


Рисунок 3.3 - Алгоритм побудови сигнатури процесу

3.2.2 Розроблення алгоритму роботи модуля, що включається до складу ядра операційної системи

Даний модуль повинен задовольняти наступним вимогам:

- він повинен повністю відповідати обмеженням наведеним на початку розділу;
- у модулі повинно бути вказано хто є його автором, тип ліцензії по якій він розповсюджується, версію та його короткий опис;
- він повинен бути скомпільований в тій же версії ядра в якій він буде використовуватися, це пов'язано з тим, що під час компіляції до нього додається інформація про версію ядра, коли він включається в ядро, то ця інформація перевіряється, якщо вона співпадає, то модуль успішно включається, в іншому ж випадку повертається повідомлення про помилку.

Алгоритм модуля, що включається до складу ядра ОС, полягає у створенні нескінченого циклу, у якому виконуються наступні дії:

Алгоритм 3.2.

3.2.1. Для перевірки обирається процес, який присутній в черзі процесів, які готові до виконання або виконується в даний момент.

3.2.2. Для цього процесу будується нова сигнатура. Відбувається збір інформації про ті ресурси, які використовує даний процес;

3.2.3. Обчислюється гранична імовірність при перевищені якої можна сказати, що процес буде заблокований іншим процесом або сам заблокує інший процес;

3.2.4. Відбувається перегляд ресурсів, які використовує даний процес і обчислення нової імовірності з врахуванням цих ресурсів;

3.2.5. Порівняння цих двох ймовірностей і у випадку, якщо остання більша відбувається спроба забирання ресурсу, який вносить найбільше значення у отриману величину імовірності. У випадку якщо остання імовірність менша, то відбувається перехід до кроку 3.2.7;

3.2.6. Відбувається спроба забрати вищеописаний ресурс у процесу, якщо вона закінчилась вдало, то перехід до кроку 3.2.7, інакше процес завершує своє виконання;

3.2.7. В файл записується детальна інформація про даний процес. Обчислюється та зберігається нова сигнатура;

3.2.8. Перевіряється час тестування, якщо модуль уже відпрацював свій час, то відбувається вихід з циклу та вивантаження його з ядра. Інакше перехід до кроку 3.2.1.

3.2.9. Кінець алгоритму

Алгоритм роботи модуля наведений на рис.3.4.

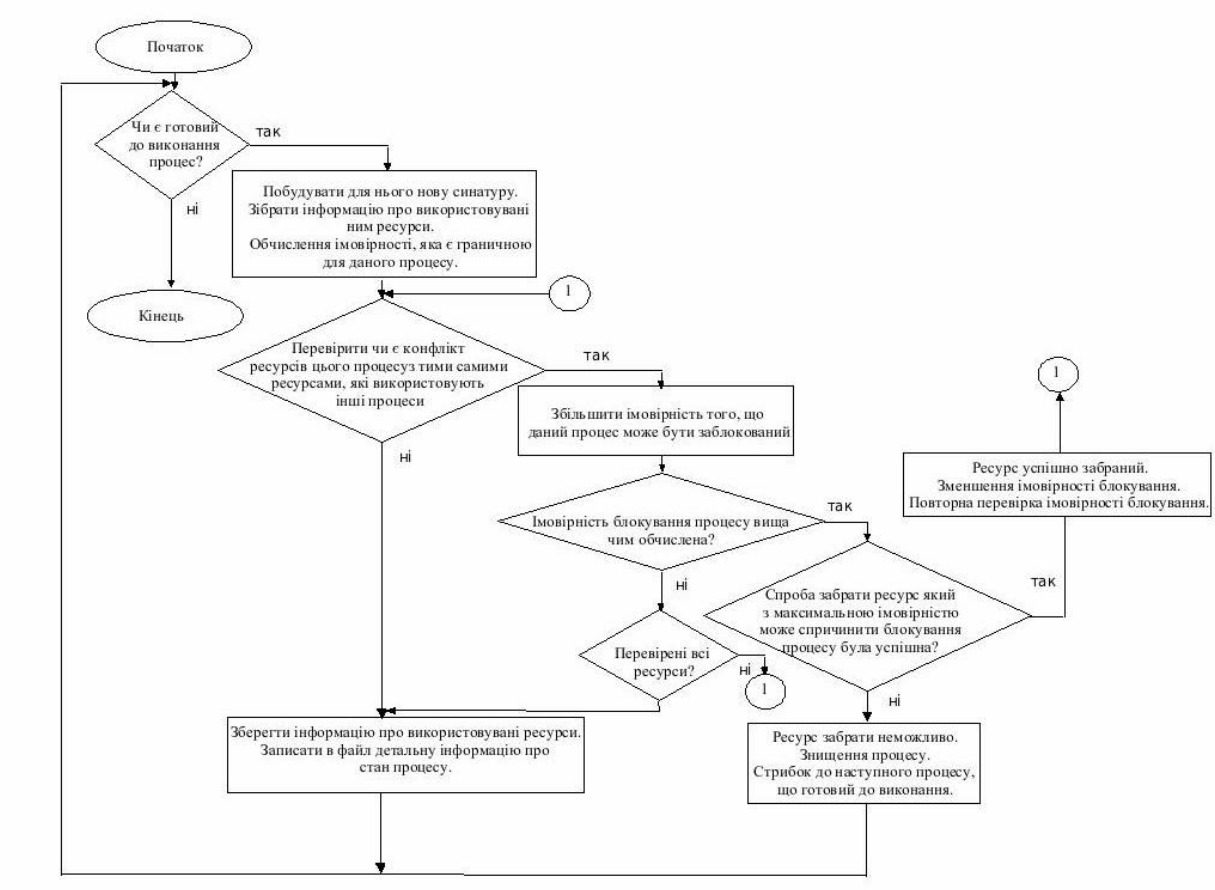


Рисунок 3.4 - Алгоритм роботи модуля ядра

3.2.3 Алгоритм обчислення ймовірності настання взаємоблокування

Оскільки кожен процес під час свого виконання використовує різноманітні ресурси, то очевидно, що алгоритм обчислення імовірності взаємоблокування повинен враховувати всі ті ресурси, які даний процес використовує. Процес може використовувати такі види ресурсів як: файли, пам'ять, ресурси апаратури, сигнали, семафори. Тому алгоритмом повинні враховуватися всі ці ресурси.

Отже, алгоритм обчислення ймовірності взаємоблокування для поточного процесу можна представити такими кроками:

Алгоритм 3.3.

3.3.1. Збір інформації про використання файлової системи;

3.3.2. Збір інформації про використання пам'яті;

3.3.3. Збір інформації про використання сигналів;

3.3.4. Збір інформації про використання семафорів;

3.3.5. Фінальне обчислення імовірності взаємоблокування шляхом сумування вищенаведеної інформації.

3.3.6. кінець алгоритму

3.3 Оцінка ефективності алгоритмів прогнозування стану.процесів

Оцінка ефективності алгоритму включає як якісний так і кількісні показники. Якісним показником є те, чи вирішує алгоритм поставлену задачу, чи ні. Кількісними показниками є час виконання та ємність алгоритму. В даному випадку критичними показниками є час, за який буде виконуватись прогнозування настання ситуації взаємоблокування, та задіяні при цьому ресурси системи. Отже, критерієм для оцінки ефективності алгоритму буде час.

Згідно загальноприйнятих підходів основним показником часової ефективності є часова складність (ЧС) – порядок складності алгоритму O(f).

Для алгоритму 1 ЧС становить O(k), де k - кількість наявних в системі процесів, оскільки k разів повторюються однакові лінійні дії.

Для алгоритму 2 ЧС становить $O\left(j \cdot \log\left(n\right)\right)$, де n – кількість процесів, що знаходяться у граничному стані, j – кількість правил, що містяться у базі правил.

На рис. 3.5 наведена ЧС алгоритму прогнозування стану процесу для різної кількості процесів та правил. Як видно із рис. 3.5, складність алгоритму зростає нелінійно при збільшенні кількості процесів у системі, оскільки, на відміну від відомих алгоритмів вирішення задачі взаємоблокування, розроблений алгоритм прогнозування стану процесів використовує компоненти нечіткої логіки, що робить його гнучким у використанні.

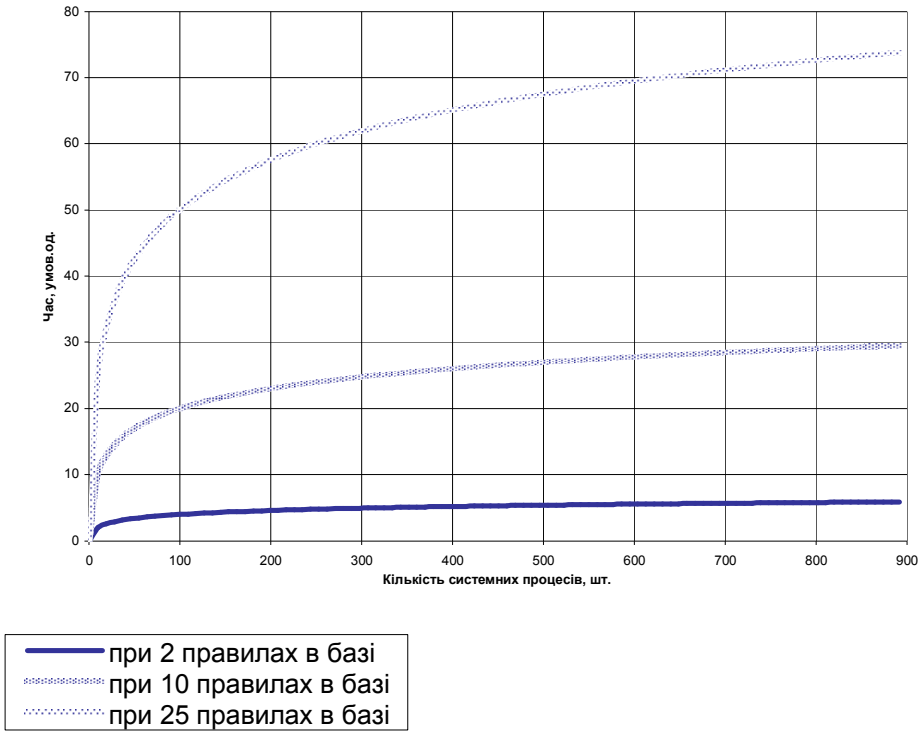


Рисунок 3.5 - Часова складність алгоритму прогнозування стану процесу

Суть використовуваного методу [41,75] в порівнянні з відомими [97,98,108] методами оцінки ефективності алгоритмів прогнозування стану процесів полягає в необхідності розв'язати дану задачу без врахування логічної структури обчислювальних засобів, на яких реалізується алгоритм у вигляді програми, а також при відсутності конкретних даних про обмеження на часткові параметри алгоритму.

Дана методика передбачає наступні початкові положення, які описують вхідні обмеження на аргументи та результати. Сукупність часткових параметрів алгоритму A_1 можна представити у вигляді множини $X_1 = \{\vec{X}_1,...,\vec{X}_k\}$, (i = 1,2,...,k). Кожен вектор може бути багатокомпонентним і позначатись у вигляді $\vec{X}_i = (x_{i1},...,x_{igi})$, де gi- число компонентів вектора $\vec{X}_i$; k-число параметрів алгоритму A_1. Виберемо таку множину X_1', яка в достатній мірі характеризує алгоритм A_1 $(X_1' \subset X_1)$. Тоді критерії визначаються у вигляді функції $F(A_1) = F[A_1(X_1')]$. Якщо є N

алгоритмів (j=1,2,...,N), то оптимізація алгоритмів за їх параметрами приводиться до знаходження досить складної функції, параметри якої найбільш задовільняють обмеженням на часткові критерії

$$(\min x_q) \le F[A_j(X'_j)] \le (\max x_q), \text{ (q=1,2,...,s)}, \qquad (3.8)$$

де s - кількість елементів множини X'_j. Мінімізація часу виконання алгоритму $\min\{\sum_p t_{p1} * y_{p1},\ldots,\sum_p t_{pN} * y_{pN}\}$, де t_{pj} - час виконання p-ої операції; y_{pj} - кількість p-го типу операцій та визначення обмежень на інші часткові параметри алгоритму ще більш ускладнюють цю задачу. Тому для спрощення розв'язку цієї задачі використовується дана методика оцінки ефективності алгоритмів. Для цього вирішено задачу оптимізації без врахування структури обчислювальних засобів, на яких реалізується алгоритм у вигляді програми, а також за відсутністю конкретних даних про обмеження на часткові параметри алгоритму. Для вирішення цієї задачі алгоритм роботи опишемо графо-аналітичною моделлю. При цьому відображення алгоритму здійснимо однозв'язною частково упорядкованою множиною M_z, а множину M_z відобразимо орієнтованим графом. В множині M_z z- має певне значення ($z < Z_0$), тобто Z_0 - кінцеве та дискретне. Елементи множини відображені вершинами на графі, а відношення між елементами множини - у вигляді направлених дуг. Кінцева дискретна впорядкована множина подана у вигляді орієнтованого графа. Крім того, згідно даної методики неоднозв’язна частково упорядкована множина розкладається на декілька упорядкованих множин. При відображенні алгоритмів на підставі властивостей відношення випередження виключимо ряд зв'язків, зокрема петлі, бо вони поглинаються за рахунок властивості рефлексивності. Цикли будемо розривати шляхом декомпозиції алгоритму на фрагменти. Таке виключення зворотного випередження надасть графу, який відображає

алгоритм, властивості антисиметричності. Несуттєві зв'язки в алгоритмі виключаються на основі властивості транзитивності.

Оптимізація алгоритму зводиться до оптимізації структури множини M_z, яка відображає алгоритм, та мінімізації загального числа елементів в ній. Геометрично процедура пошуку відображається на бінарному дереві графа відповідного алгоритму. Форма бінарного дерева може істотно змінюватися при заданому числі одиничних виборів n та числі наслідків одиничного вибору m. Залежно від форми бінарного дерева змінюються його кількісні характеристики, зокрема такі, як довжина окремих гілок (довжина бінарного дерева). Цей параметр бінарного дерева відображає один із часткових кількісних параметрів для алгоритму, що оптимізується. Довжина бінарного дерева L визначиться сумарною довжиною його гілок, у яких немає повторень однойменних елементів, тобто

$$L(n,m)=\sum_{i=1}^{s} L_i, \tag{3.9}$$

де L_i - довжина і-ї гілки бінарного дерева; s- число гілок бінарного дерева, в яких немає повторень окремих елементів.

За частковий критерій ефективності $k_{Нb}$, виходячи з декомпозиції бінарного дерева, прийнято відношення довжини гілки дерева $L_{i(\epsilon.\blacklozenge.)}$, яка відображає граф, до всієї довжини L бінарного дерева. Тоді частковий коефіцієнт ефективності:

$$k_{Нb(L)}=\frac{L_{i(\epsilon.\blacklozenge.)}}{L}. \tag{3.10}$$

Оптимізованим в такому випадку буде алгоритм, для якого $k_e \to 1$ при $0< k_e \le 1$. При порівнянні алгоритмів більш оптимальним буде алгоритм, у якого більший коефіцієнт $k_{e(L)}$.

Довжина $L_{i(\epsilon.\blacklozenge.)}$ знаходиться шляхом декомпозиції підграфа дерева на підграфи без повторення в них однойменних елементів.

Для підрахунку числа циклів в алгоритмі введено поняття уточненого часткового коефіцієнта ефективності $k_{e.—(L)}$. Уточнення полягає в тому, що для даного випадку при обчисленні $k_{э.—(L)}$ довжина бінарного дерева L буде замінюватися деякою абстрактною довжиною L_a, яка буде обчислюватися за формулою:

$$L_a = L + \sum_{i=1}^{r} n_i * L_{i(ль)}, \tag{3.11}$$

де r - кількість циклів; n_i - кількість повторень і-го цикла; $L_{i(ль)}$ - довжина і-го цикла. Тоді

$$k_{e.—(L)} = \frac{L_{i(\epsilon.\blacklozenge.)}}{L_{\epsilon}}. \tag{3.12}$$

Введений частковий коефіцієнт ефективності $k_{e(L)}$ ($k_{e.—(L)}$) характеризує ефективність алгоритмів, які розглядаються з врахуванням довжини бінарних дерев, відображаючих їх. Крім того, одним із істотних параметрів, що кількісно описують алгоритм, є час t (ємність) виконання операторів алгоритму. Він дорівнює:

$$t = \sum_{i=1}^{m} t_i, \tag{3.13}$$

де t_i - час виконання і- го оператора; m- число операторів в алгоритмі.

Оцінка ефективності алгоритму з урахуванням часу виконання операторів не завжди зручна за причиною зв'язку часу виконання операторів з конкретними апаратними засобами. Зручніше оцінювати алгоритми з точки зору ефективності, тому й введено коефіцієнти складності виконання операторів.

Для керуючих алгоритмів та відповідних їм програм скористуємося ваговими коефіцієнтами регресії (складність виконання операторів) α.

В загальну кількісну оцінку ефективності алгоритму включимо середній коефіцієнт складності виконання операторів

$$\alpha_{-\text{p}} = \frac{\sum_{i=1}^{p} \alpha_i}{p}, \qquad (3.14)$$

де α_i- коефіцієнт складності виконання і- го оператора; р- число операторів в алгоритмі.

Узагальнений коефіцієнт ефективності алгоритму k_e з врахуванням двох незалежних часткових коефіцієнтів визначається як їх добуток: $k_e = k_{e.—(L)} * (1 - \alpha_{-\text{p}})$. Позначимо $k_{e.—(L)} = k_1$; $1 - \alpha_{c\,\text{p}} = k_2$. Тоді $k_e = k_1 * k_2$.

Якщо часткових коефіцієнтів буде q, то узагальнений коефіцієнт ефективності алгоритму k_e розраховується за наступною формулою:

$$k_e = \prod_{i=1}^{q} k_i, \qquad (3.15)$$

де k_i - і - й частковий коефіцієнт ефективності алгоритму.

Уточнений частковий коефіцієнт ефективності зменшується із збільшенням числа циклів в алгоритмі, а також із збільшенням числа повторів циклів алгоритму за експоненціальним законом.

Згідно використовуваної методики узагальнений коефіцієнт ефективності алгоритму $k_e = 0.78$.

Отримані розрахунки ефективності алгоритму є задовільними для практичної реалізації прогнозування стану процесів в персональному комп'ютері.

3.4 Оцінка часової складності методу прогнозування взаємоблокування процесів

Для оцінки часової складності методу прогнозування взаємоблокувань процесів було використано web-сервер Apache2 з PHP 5

та sql-сервер MySQL. Для перевірки роботи методу на сервері запускався на виконання PHP-скрипт, представлений у лістингу 1, який при паралельному виконанні призводить до виникнення взаємоблокувань.

Лістинг 1

```
$sql = "SELECT COUNT(*) FROM t "; $x = mysql_query($sql);
$r = mysql_fetch_row($x);$max_id = $r[0];
$id1 = rand(0,$max_id); do { $id2 = rand(0,$max_id); } while ($id1==$id2);
mysql_query("START TRANSACTION;");
$sql1 = "select a from t where id = $id1 for update"; mysql_query($sql1);
//Вибір кортежу для обрахунку
usleep(100); //моделювання обробки даних
$sql2 = "select b from t where id = $id2 for update"; mysql_query($sql2);
mysql_query("COMMIT;"))
```

В представленому коді видалено бізнес-логіку, зате повністю збережено послідовність запитів, що при паралельному виконанні призводять до появи взаємних блокувань.

Рівні взаємоблокувань, представлені на рис.3.6, показують взаємоблокування в КС. Суцільною лінією показано рівень взаємоблокувань при використанні стандартних засобів MySQL для виявлення заблокованих транзакцій. Пунктирною лінією показано рівень взаємоблокувань, що виникали при використанні запропонованого методу прогнозування взаємоблокувань.

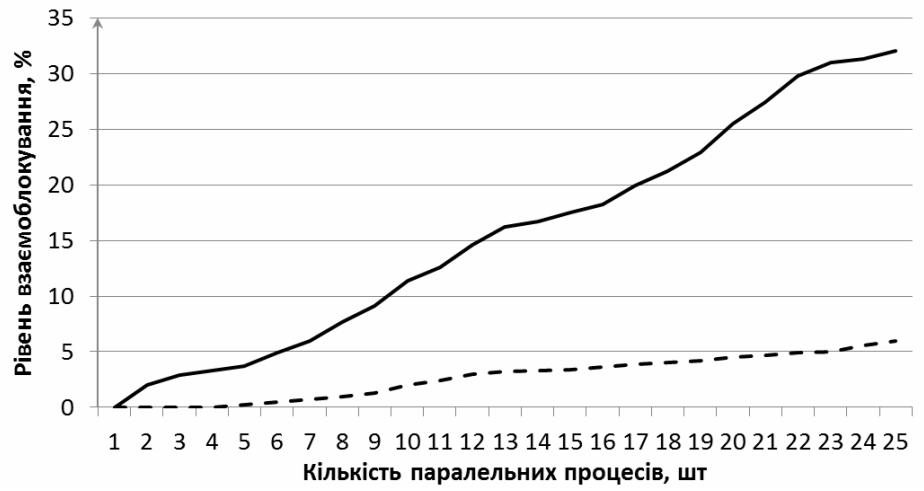


Рисунок 3.6 - Залежність рівня взаємоблокувань від кількості паралельних процесів

Час виконання процесів при використанні різних механізмів вирішення взаємоблокувань показаний на рис.3.7. Суцільною лінією показано середній час виконання процесів при використанні стандартного механізму вирішення взаємоблокувань. Пунктирною – середній час виконання процесів при використанні запропонованого методу прогнозування взаємоблокувань. Штрих-пунктирною – середній час виконання процесів, за умови ненастання взаємоблокувань.

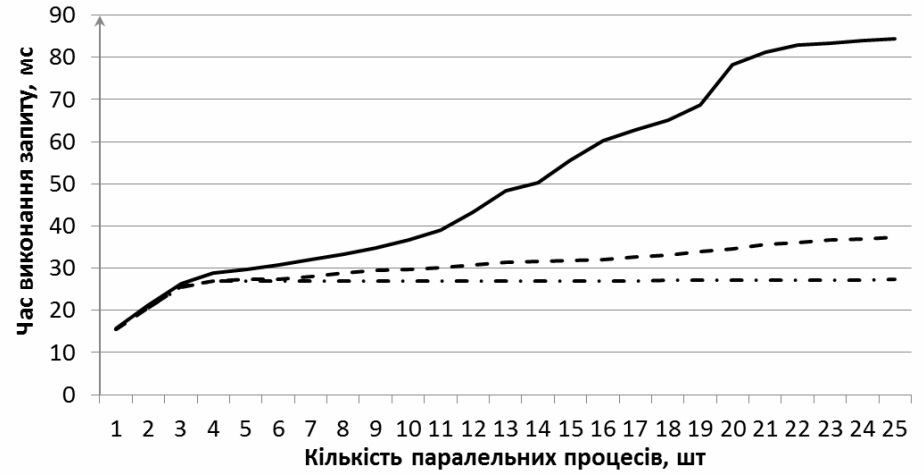


Рис. 3.7 - Залежність часу обробки запитів від кількості паралельних процесів

У випадку використання стандартного механізму виявлення взаємоблокувань спостерігається значне зростання часу виконання навіть при незначній кількості паралельних процесів. В цей самий період не спостерігається значного завантаження процесора (не більше 20%), оскільки процеси більшу частину часу очікують доступу до заблокованих ресурсів.

У випадку, коли не виникає взаємоблокувань процесів, час їх виконання зростає незначно (на 3% при кількості паралельних процесів 25 і рівні завантаження процесора не більше 20%).

Використання запропонованого методу прогнозування взаємоблокувань показує помітно пологіше наростання середнього часу виконання процесів, за таких самих умов.

Зниження часу виконання процесів при виконанні єдиного потоку пояснюється відсутністю затримок, пов'язаних з очікуванням звільнення ресурсу.

В ході дослідження було проведено порівняння часу виконання фаз транзакцій при максимальному завантаженні системи і різних режимах роботи, що представлено на рис.3.8.

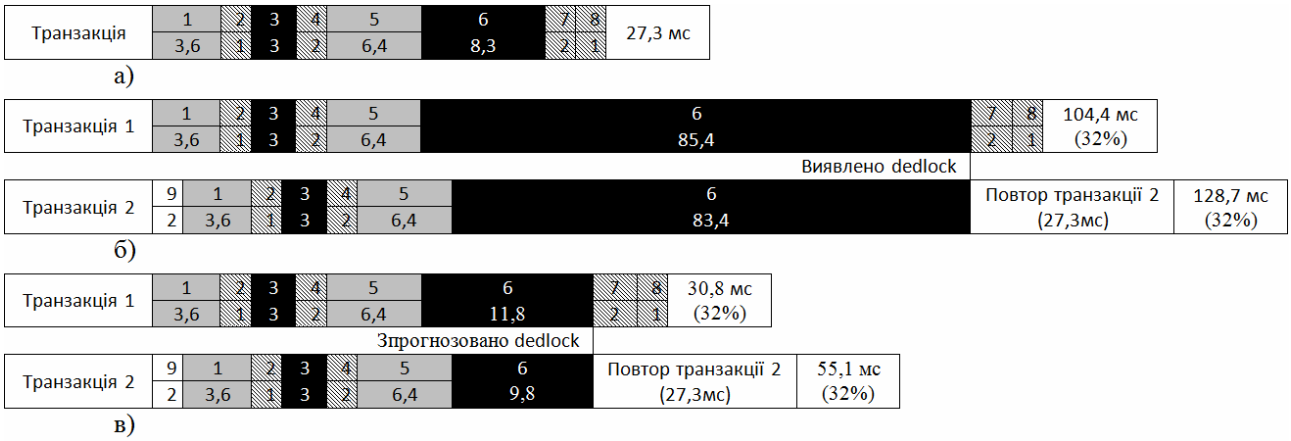


Рис. 3.8 - Часові діаграми виконання фаз транзакції:

а) без взаємоблокування; б) взаємоблокування із стандартним виявленням; в) взаємоблокування із прогнозуванням. (1 – Підготовчі дії; 2 – Запуск транзакції; 3 – Очікування звільнення кортежу id1; 4 – Виконання 1-го

SELECT; 5 – Опрацювання даних; 6 – Очікування звільнення кортежу id2; 7 – Виконання 2-го SELECT; 8 – Завершення транзакції; 9 – Затримка від початку запуску.)

На рис. 3.8а представлена часова діаграма виконання фаз транзакції за умови, що при її виконанні не відбулось взаємоблокування. Середній час виконання транзакції склав 27,3мс, з них 11,3мс (41%) зайняло очікування заблокованих ресурсів. В ході дослідження таких транзакцій виявилось 36%.

На рис. 3.8б представлено часову діаграму взаємоблокування двох транзакцій із стандартним механізмом їх вирішення. Транзакція 1 при першому звертанні (4) до таблиці БД вибирає та блокує кортеж з ключем id1. Транзакція 2 розпочинає виконуватись на 2мс пізніше, ніж Транзакція 1 і при першому звертанні (4) вибирає та блокує запис із ключем id2. Після опрацювання отриманих даних (5) транзакція 1 звертається до кортежу з id2. Оскільки він заблокований, то транзакція переходить до очікування звільнення ресурсу (6). В свою чергу транзакція 2 звертається до кортежу з id1. Він також виявляється заблокованим і транзакція 2 переходить в режим очікування звільнення ресурсу (6). Процеси потрапляють в ситуацію взаємного блокування і самостійно не можуть вийти з циклу нескінченного очікування. Взаємоблокування вирішує КС періодично аналізуючи граф процесів і ресурсів. Оскільки задача є алгоритмічно складною, то вона виконується через інтервали часу в 150-200мс. Після виявлення взаємоблокування один із процесів, що пізніше надійшов у систему, примусово завершується і повторно запускається на виконання. Інший з заблокованих процесів продовжує своє виконання. Великі часові проміжки між повторними аналізами графа призводять до значних затримок при виявленні взаємоблокувань. Середній час процесу, який був виконаний повторно складає 128,7мс, процесу, який продовжив роботу після взаємного

блокування – 104,4мс. В ході дослідження таких транзакцій виявилось по 32%, оскільки процеси у взаємоблокування потрапляють парами.

На рис. 3.8в представлено часову діаграму взаємоблокування двох транзакцій із прогнозуванням взаємоблокувань. До моменту запиту в транзакції 2 кортежу з ключем id2 часові діаграми ідентичні. В момент цього запиту (6) транзакція 2 потрапляє в граничний стан і для неї проводиться прогнозування. Час прогнозування складає близько 10мс. В результаті прогнозу визначається ймовірність взаємоблокування процесів, після чого обирається один із них. Цей процес знімається з виконання і запускається повторно. Середній час виконання процесу, що був виконаний повторно, складає 55,1мс; процесу, що продовжив роботу після взаємного блокування – 30,8мс. В ході дослідження таких транзакцій виявилось по 32%.

З отриманих досліджень середній час виконання транзакцій із стандартним механізмом виявлення взаємоблокувань склав: $27,3 \cdot 0,36 + 104,4 \cdot 0,32 + 128,7 \cdot 0,32 = 84,4$ мс, із прогнозуванням взаємоблокувань: $27,3 \cdot 0,36 + 30,8 \cdot 0,32 + 55,1 \cdot 0,32 = 37,3$ мс.

Застосування методу прогнозування взаємоблокувань забезпечило зменшення часу виконання процесів в $\frac{84,4}{37,3} = 2,3$ рази. Це дозволяє більш ефективно використовувати ресурси КС.

3.5 Висновки

Розроблено метод прогнозування взаємоблоекувань процесів в КС, в основі якого лежить сигнатурна модель процесу та модель прогнозування взаємоблокування процесів.

Запропоновано оцінку часової складності та ефективності методу прогнозування взаємоблокувань процесів в КС. Проведено його експериментальне дослідження на прикладі модифікованої СКБД MySQL. Середній час виконання процесу зменшився з 84,4мс при використанні

стандартних засобів MySQL до 37,3 мс при використанні запропонованого методу. Отримані результати вказують на зменшення часових витрат на виконання процесів, що в свою чергу дозволяє опрацьовувати в 2,3 рази більшу кількість даних в одиницю часу для задач, в яких часто виникають взаємоблокування (відбувається конкурентна боротьба за ресурси).

4 РОЗРОБЛЕННЯ ПРОГРАМНИХ ЗАСОБІВ ПРОГНОЗУВАННЯ СТАНУ ПРОЦЕСІВ В ПЕРСОНАЛЬНОМУ КОМП'ЮТЕРІ

4.1 Програмні засоби визначення зміни стану процесу

4.1.1 Опис зв’язків між ПЗ та ОС

При розробці програмного забезпечення для створення сигнатур процесів що виконуються, використовувались бібліотечні функції, системні виклики та системні завдання.

У ОС Minix3 використовувати системні завдання мають право тільки процеси системного рівня (драйвери, сервери). Тому, для виконання поставленої задачі було створено спеціальний сервер що має доступ до потрібних нам системних структур та функцій.

Дана ОС має багато особливостей. Одне з яких – де централізованість системних структур даних. Так, при мікроядерній архітектурі, у ядрі міститься лише обмежений набір функцій для роботи з апаратною складовою, підтримки преривань, обміну повідомленнями, частина задач планування процесів, перемикання контексту, забезпечення роботи системного годинника та деяких інших.

Більшу частину роботи по забезпеченню виконання користувацьких процесів виконують спеціальні сервери та драйвери що працюють по за ядром, у користувацькому просторі. Але, при цьому, вони мають значно ширші права на передачу повідомлень та спілкуються з ядром через спеціально розроблені інтерфейси.

Таким чином, щоб отримати найбільш повну інформацію про стан процесу, потрібно, окрім ядра, опитувати відповідні сервери, що відповідають за певний інтерфейс яким користуються процеси.

Головну роль у контролі між серверної взаємодії відіграє сервер реінкарнації. Його робота полягає у тому, щоб слідкувати за життєвим циклом сервера та дотриманням правил доступу до системних функцій

ядра. Правила доступу вказуються у системному файлі конфігурації серверів /etc/system.conf.

Нижче подано схему зв’язків розроблюваного сервера з іншими серверами які беруть участь у формуванні сигнатури кожного активного процесу (рис. 4.1).

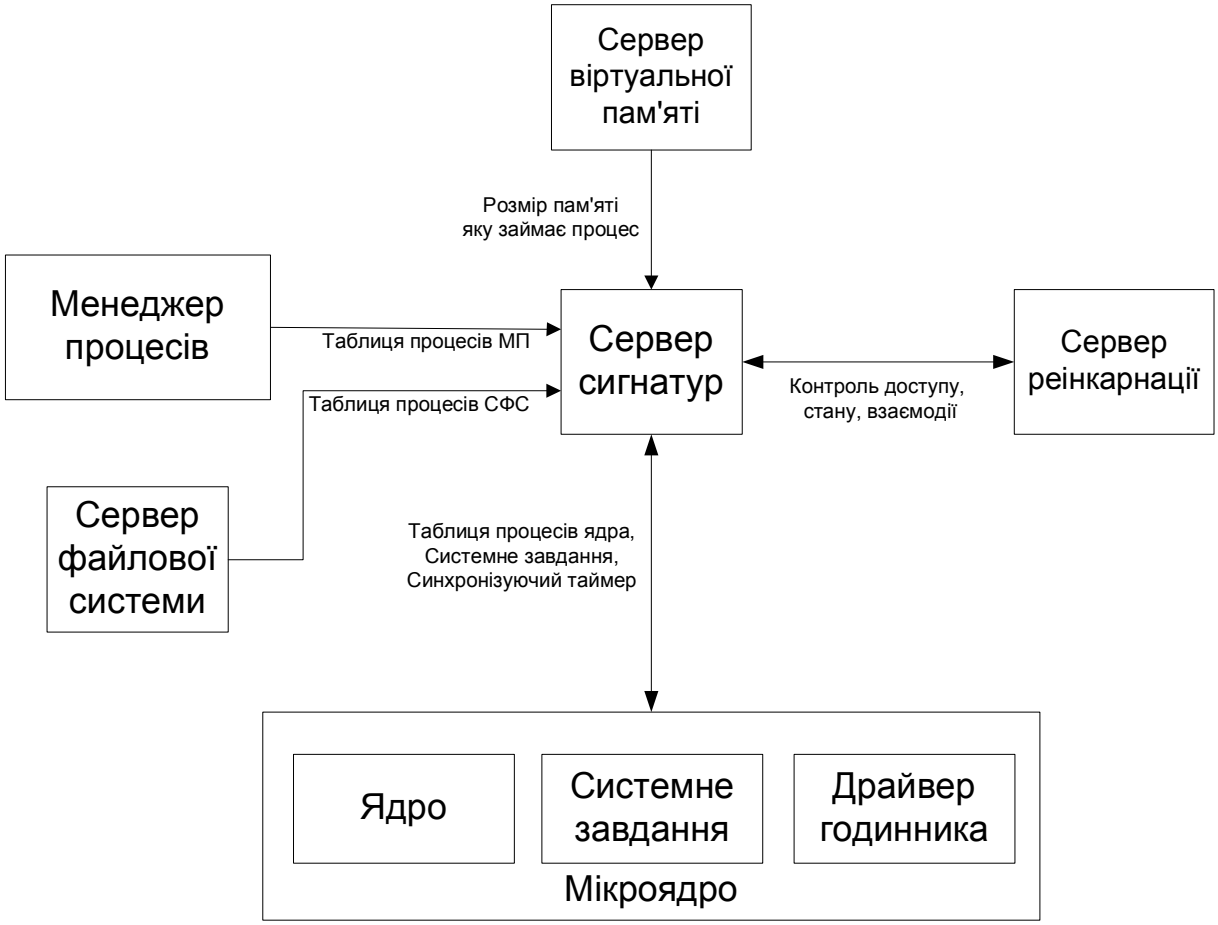


Рисунок 4.1 - Зв’язки ПЗ з іншими компонентами ОС

Як бачимо з наведеного рисунка, інформація про активні процеси витягається з таблиць процесів ядра, менеджера процесів, сервера файлової системи, сервера віртуальної пам’яті. Окрім таблиці процесів, ядро надає послуги системного годинника, який дозволяє використовувати сторожовий таймер для синхронізації операцій створення сигнатур процесів.

Також сервер сигнатур не явно користується послугами сервера системного журналу, що дозволяє записувати діагностичні повідомлення, наприклад, у випадку одержання невідомого повідомлення.

4.1.2 Опис алгоритму

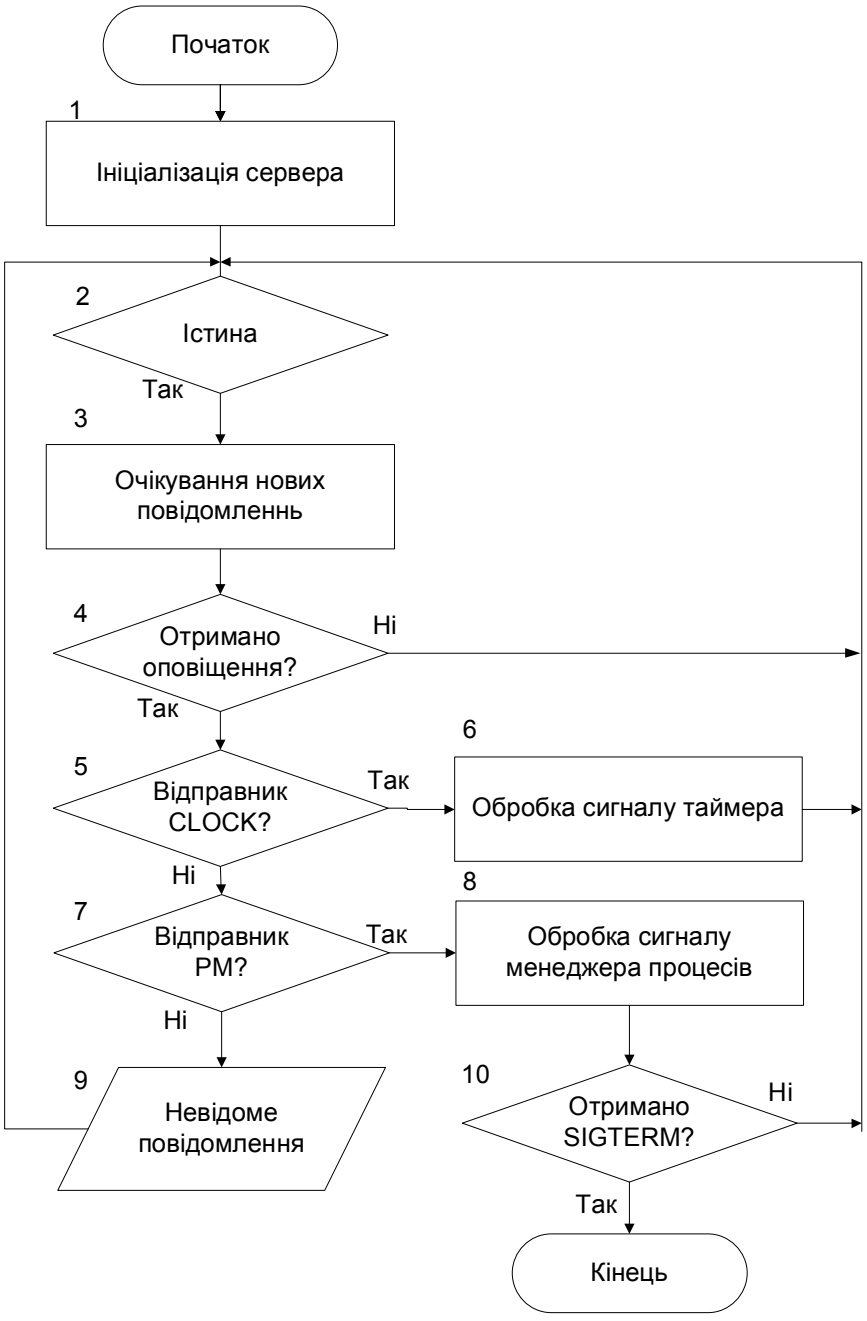


Рисунок 4.2 - Укрупнена блок-схема алгоритму програми

Блок 1. Виконуються специфічні для ОС Minix3 команди по реєстрації нового сервера серед системних процесів, при цьому

використовується бібліотека системних подій (SEF). Процес реєстрації включає в себе заповнення відповідних структур сервера реінкарнації (RS) що дозволить йому виконувати початкову ініціалізацію сервера та подальше обслуговування при необхідності перезапуску чи обновлення. Також при виконанні ініціалізації відбувається створення каналу для виводу інформації, визначення частоти системного годинника, початковий збір сигнатур усіх процесів та їх вивід, встановлення синхронізуючого таймера.

Блок 2. Умова що формує безкінечний цикл. Є основую будь якого сервера чи драйвера. Використовується для безперервного отримання повідомлень та їх обробки.

Блок 3. Виконується блокуючий виклик процедури отримання повідомлення. При надходженні повідомлення у відповідних глобальних змінних зазначається тип повідомлення та його джерело.

Блок 4. Перевірка типу отриманого повідомлення. Оповіщення?

Блок 5. Умова визначення джерела повідомлення. Істина, при отриманні оповіщення від системного драйверу годинника.

Блок 6. Обробник сигналу таймера. Виконуються основні дії для роботи монітору процесів: створення дампу сигнатур усіх процесів на поточний момент, пошук процесів у сигнатурі яких відбулися зміни у порівнянні з попереднім дампом, вивід знайдених сигнатур у вихідний потік, значення попередніх дампів замінюються новими, планується новий сигнал синхронізуючого таймера.

Блок 7. Умова визначення джерела повідомлення. Істина, при отриманні оповіщення від системного менеджера процесів.

Блок 8. Обробка сигналу менеджера процесів відбувається шляхом визначення активних сигналів POSIX.

Блок 10. При отриманні сигналу SIGTERM – Вихід.

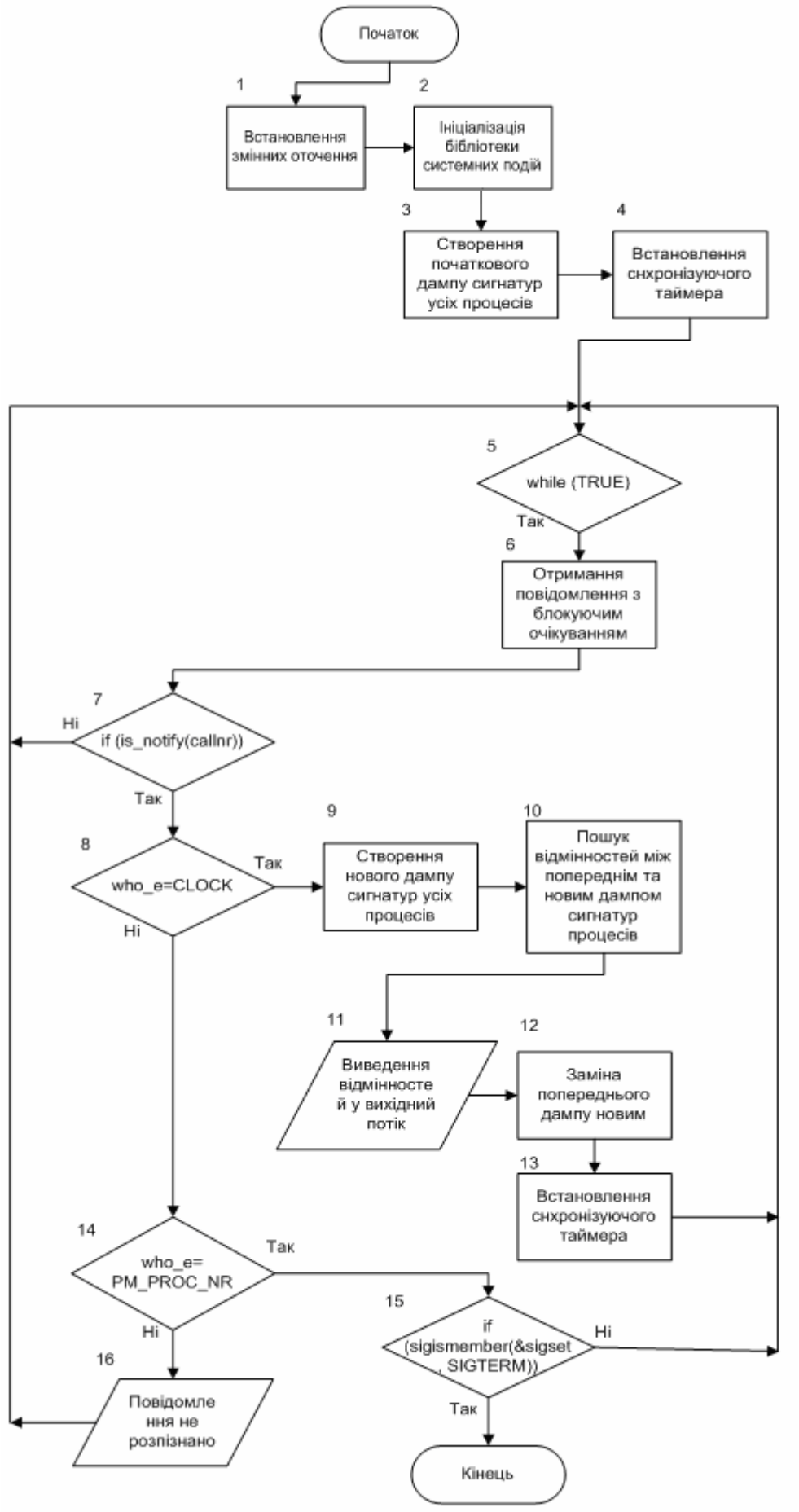


Рисунок 4.3 - Детальна блок-схема алгоритму програми

Блок 1. Встановлення змінних оточення переданих програмі під час запуску з командної стічки.

Блок 2. Ініціалізація бібліотеки системних подій (System Event Framework). Кожен системний сервіс (сервер або драйвер пристрою), повинні викликати sef_startup () при завантаженні системи для обробки ініціалізації та sef_receive () при отриманні повідомлення.

Блок 3. Створення початкового дампу сигнатур усіх процесів та виведення їх у вихідний потік. У подальшому початковий дамп використовується для знаходження відмінностей між сигнатурами у наступних дампах.

Блок 4. Встановлення синхронізуючого таймера. Синхронізуючий таймер дає можливість виконувати створення нових дам пів сигнатур через вставлений період часу.

Блок 5. Умова що формує безкінечний цикл. Є основую будь якого сервера чи драйвера. Використовується для безперервного отримання повідомлень та їх обробки.

Блок 6. Виконується блокуючий виклик процедури отримання повідомлення sef_receive (). При надходженні повідомлення у відповідних глобальних змінних зазначається тип повідомлення та його джерело.

Блок 7. Перевірка чи є отримане повідомлення оповіщенням. Для роботи сервера важливий сам факт оповіщення. При потребі можливо отримати додаткові данні від відправника.

Блок 8. Умова визначення джерела повідомлення. Істина, при отриманні оповіщення від системного драйверу годинника.

Блок 9. Створення нового дампу сигнатур для усіх процесів. Витягнення інформації відбувається шляхом виконання спеціальних системних викликів та системних задач що дозволяють скопіювати складові таблиці процесів що розподілена по декількох системних модулях.

Блок 10. Пошук відмінностей між попереднім та новим дампом сигнатур процесів. Порівнюються сигнатури двох останніх дампів для кожного процесу.

Блок 11. Виведення відмінностей у вихідний потік. Для зменшення об’ємів інформації що виводиться, у вихідний потік записуються тільки сигнатури тих процесів що зазнали змін.

Блок 12. Заміна попереднього дампу новим. Так ми можемо відслідковувати зміни процесів при наступних дампах.

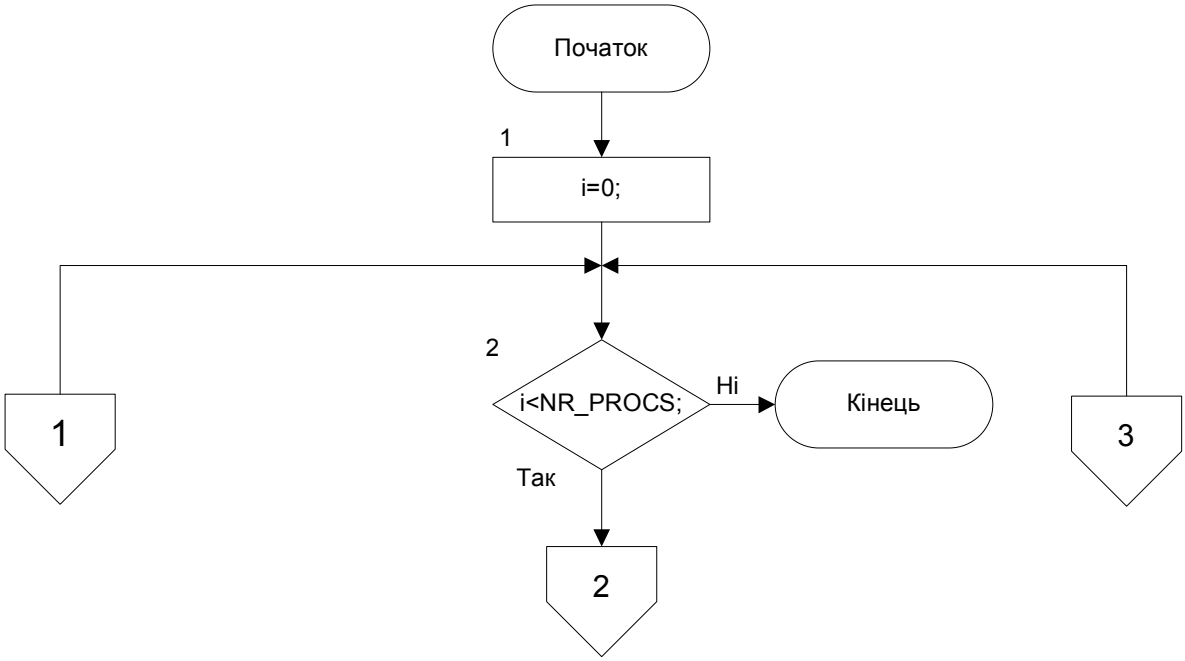
Блок 13. Встановлення синхронізуючого таймера. Синхронізуючий таймер дає можливість виконувати створення нових дам пів сигнатур через вставлений період часу.

Блок 14. Умова визначення джерела повідомлення. Істина, при отриманні оповіщення від системного менеджера процесів.

Блок 15. Обробка сигналу менеджера процесів відбувається шляхом визначення активних сигналів POSIX. При отриманні сигналу SIGTERM, робота сервера завершується.

Блок 16. Вивід оповіщення про невідоме повідомлення.

4.1.3 Опис додаткових процедур програми



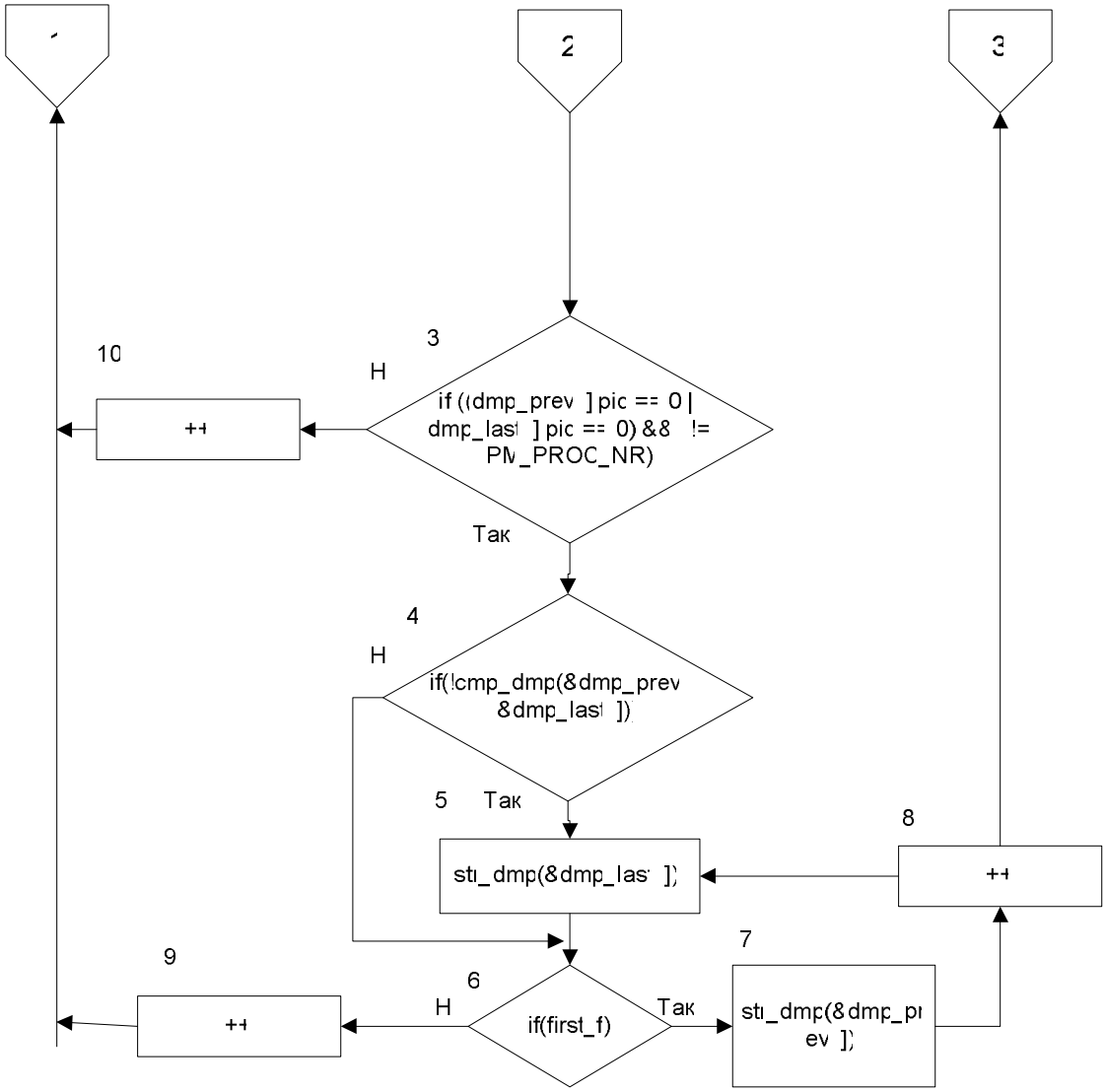


Рисунок 4.4 - Блок-схема алгоритму процедури dmp_out

Блок 1. Обнулення лічильника.

Блок 2. Перевірка перевищення максимального числа процесів.

Блок 3. Перевірка ідентифікаторів процесів.

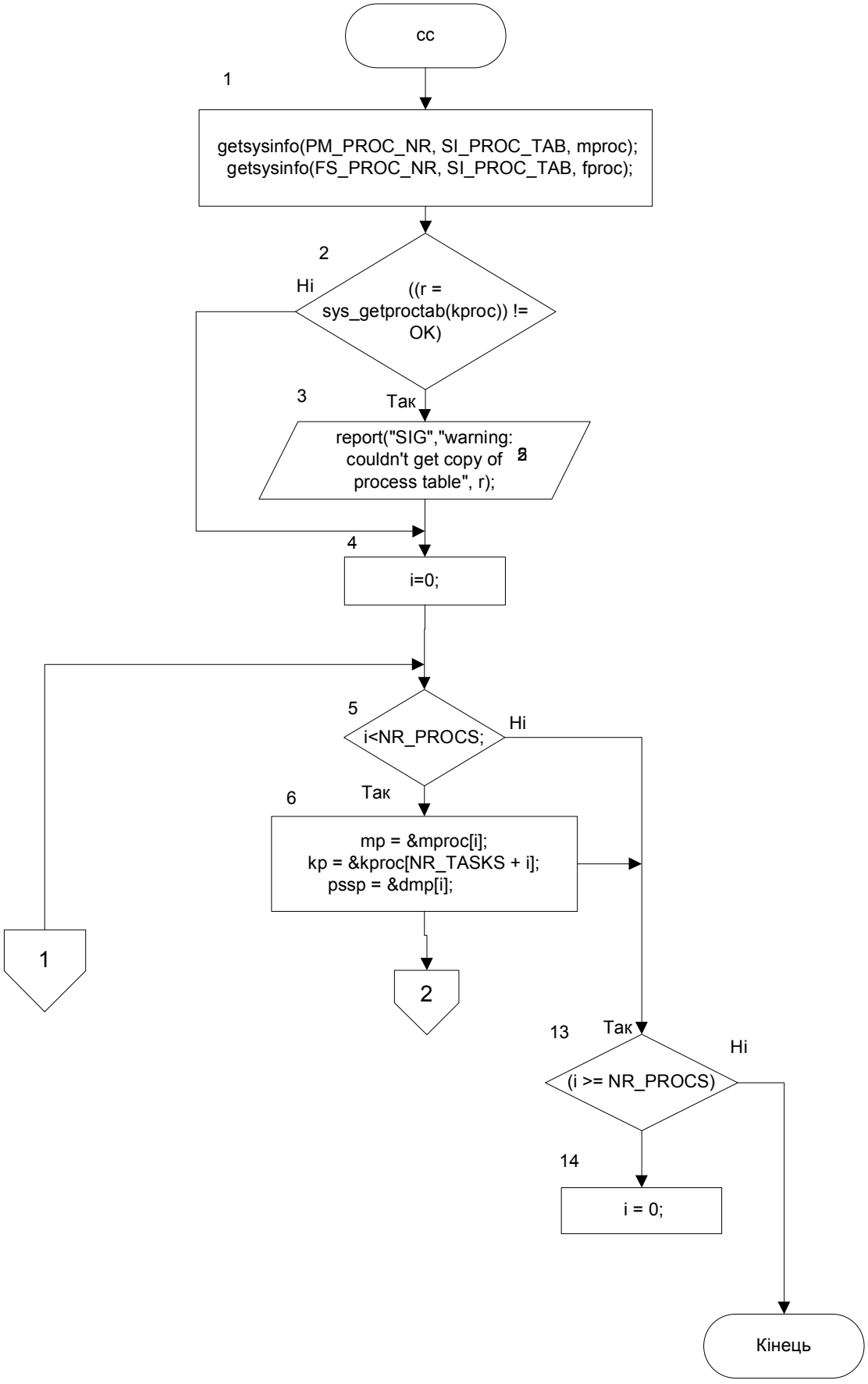
Блок 4. Порівняння сигнатур двох процесів.

Блок 5. Запис сигнатури взятої з останнього дампу у вихідний потік.

Блок 6. Перевірка установки прапору «виводити завжди».

Блок 7. Запис сигнатури взятої з попереднього дампу у вихідний потік.

Блоки 8,9,10. Збільшення лічильника на 1.


cc
1
getsysinfo(PM_PROC_NR, SI_PROC_TAB, mproc);
getsysinfo(FS_PROC_NR, SI_PROC_TAB, fproc);
2
Ні
((r = sys_getproctab(kproc)) != OK)
3
Так
report("SIG","warning: couldn't get copy of process table", r);
4
i=0;
5
i<NR_PROCS;
Ні
6
Так
mp = &mproc[i];
kp = &kproc[NR_TASKS + i];
pssp = &dmp[i];
1
2
13
Так
Ні
(i >= NR_PROCS)
14
i = 0;
Кінець

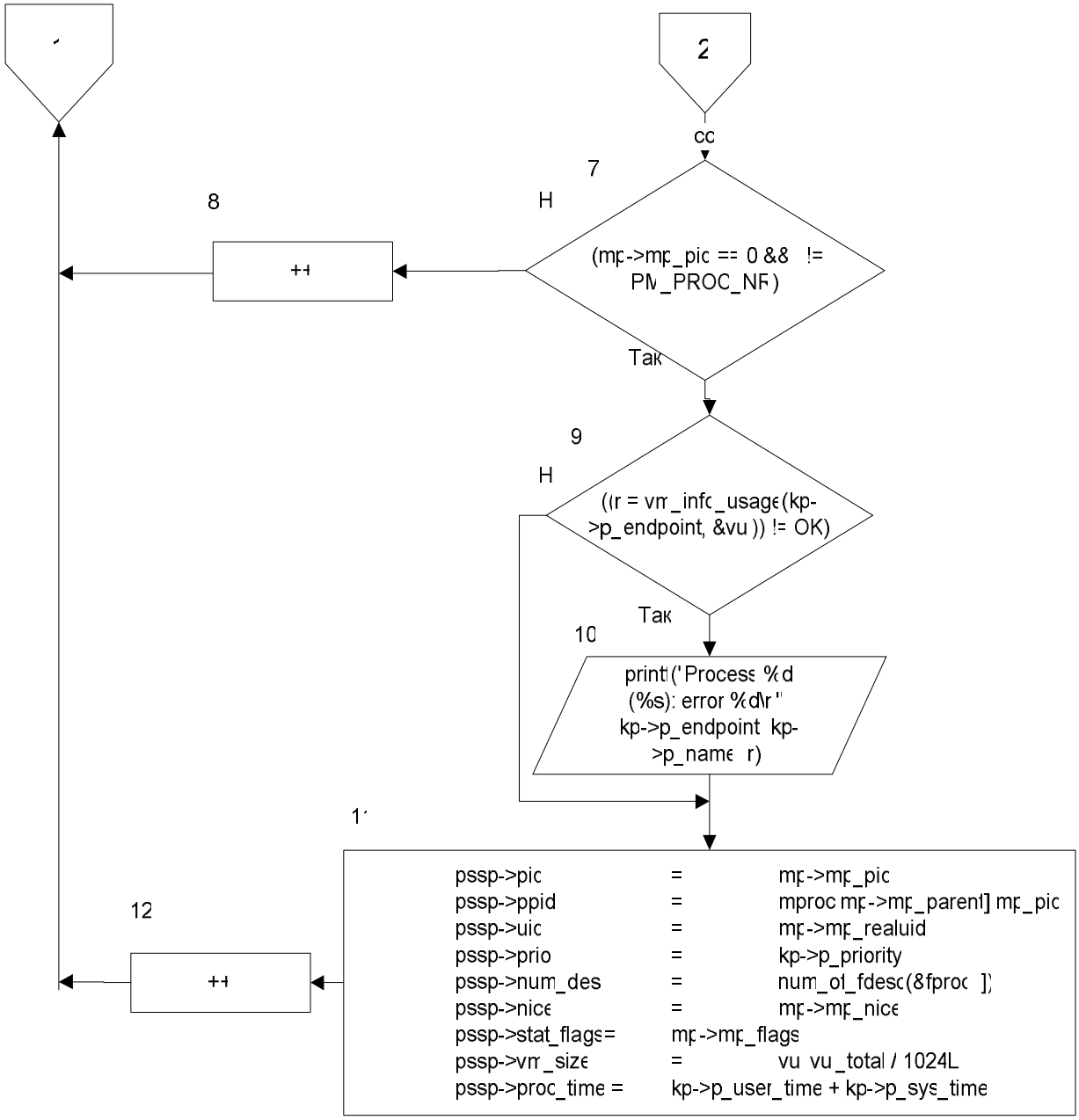


Рисунок 4.5 - Блок-схема алгоритму процедури procs_dmp

Блок 1. За допомогою системних викликів копіюємо у локальні змінні таблиці процесів менеджера процесів та сервера файлової системи.

Блок 2. Перевірка вірності копіювання у локальну змінну таблиці процесів що міститься у адресному просторі ядра. Використовується спеціальне системне завдання.

Блок 3. Запис у журнал повідомлення про не можливість скопіювати таблицю процесів ядра.

Блок 4,14. Обнулення лічильника.

Блок 5. Перевірка перевищення максимального числа процесів.

Блок 6. Присвоєння вказівникам адрес відповідних комірок таблиці процесів.

Блок 7. Перевірка ідентифікаторів процесів.

Блок 8,12. Збільшення лічильника на 1.

Блок 9. Перевірка вірності копіювання таблиці що містить данні про пам'ять що використовується процесом.

Блок 10. Виведення повідомлення про помилку процесу копіювання даних про віртуальну пам'ять.

Блок 11. Присвоєння значень елементам структури що формує сигнатуру кожного процесу.

Блок 13. Перевірка лічильника на відповідність допустимим значенням.

Інші функції та їхні параметри описані в табл. 4.1.

Таблиця 4.1 - Опис функцій

Назва процедури	Вхідний тип та змінні	Вихідний тип	Опис процедури
1	2	3	4
num_of_fdesc	struct fproc*	Int	Знаходить та підраховує активні дескриптори файлів. Повертає число відкритих дескрипторів.
procs_dmp	struct proc_stat_sign *dmp	void	Шляхом копіювання даних з системних структур, створює сигнатуру стану процесу у даний момент часу
dmp_out	struct proc_stat_sign *dmp_prev, struct proc_stat_sign *dmp_last, int first_f	void	Виконує порівняння сигнатур та запис значень у файл, фільтрує однакові сигнатури
cmp_dmp	struct proc_stat_sign *prev, struct proc_stat_sign *last	Int	По елементно порівнює значення двох сигнатур. Повертає 1 при рівності.

Продовження таблиці 4.1

1	2	3	4
str_dmp	struct proc_stat_sign *dmp	void	Виконує запис у файл вхідної сигнатури.
mix_dmp	Struct proc_stat_sign *prev,struct proc_stat_sign *last	void	Зберігає значення нового знімку сигнатур у якості старого знімку для можливості порівняння
exit_server	Void	void	Завершує роботу сервера.
sig_handler	Void	void	Визначає тип сигналу і виконує його обробку.
get_work	message *m_ptr	void	Виконує прийом повідомлень з блокуванням. Визначає тип та джерело повідомлення.
timer_handler	Void	void	Створює сигнатури процесів у момент отримання повідомлення від системного годинника та планує новий таймер.

4.1.4 Опис роботи програми

4.1.4.1 Налаштування, компіляція та встановлення сервера

Для успішної роботи розроблюваного ПЗ потрібно налаштувати систему шляхом коригування конфігураційних файлів системи. Це необхідно робити, тому що, система Minix3 висуває жорсткі обмеження що до використання спеціальних викликів ядра її ж сервісами. Таким чином забезпечується захист системи від можливих деструктивних дій стороннього ПЗ та привноситься логічне розподілення ролей між сервісами чи драйверами.

Шляхом коригування конфігураційного файлу що містить права доступу для сервісів та драйверів можливо дозволити роботу з окремими викликами системних задач що обслуговуються безпосередньо ядром, дозволити роботу з окремими інтерфейсами, шинами, апаратними

модулями, дозволити обмін повідомленнями з системними сервісами, призначити UID для процесу сервісу чи драйвера та інше.

Приклад налаштування сервера сигнатур в файлі /etc/system.conf

```
service sig
{
        system
                UMAP            # 14
                VIRCOPY         # 15
                SETALARM        # 24
                TIMES           # 25
                GETINFO         # 26
                SAFECOPYFROM        # 31
                SAFECOPYTO      # 32
                SETGRANT        # 34
                PROFBUF         # 38
                SYSCTL          # 44;
        ipc
                SYSTEM
                PM
                RS
                LOG
                TTY
                DS
                VM
                USER;
        vm
                INFO;
        uid     0;};
```

Після збереження змін потрібно скопіювати каталог з вихідними файлами та Makefile у каталог /usr/src/servers/ і відкорегувати головний Makefile каталогу servers вписавши шлях до потрібної директорії.

Для першої компіляції вихідних файлів сервера потрібна перекомпіляція усього ядра системи разом з іншими бібліотеками, сервісами та драйверами.

Виконується наступна послідовність команд:

$ su

cd /usr/src/tools

make depend

make fresh

make libraries install

Після закінчення компіляції та встановлення, в каталозі /boot/image/ буде створено знімок нового ядра системи. По замовчуванню виконуваний файл самого сервера створюється за адресою /sbin/sig.

Щоб завантажити нове ядро потрібно вказати шлях до його зображення знаходячись у моніторі завантаження.

Наприклад:

>image=/boot/image/3.1.6r1

>boot

Після повного завантаження ОС можна запускати на виконання новий сервер.

4.1.4.2 Запуск сервера

Щоб запустити на виконання розроблений сервер потрібно виконати таку команду:

#service up /sbin/sig

Як тільки сервер запуститься, він створить вихідний канал fifo /var/sig і почне вивід сигнатур. Переглянути вивід у реальному часі можна командою:

#cat /var/sig

При цьому, як тільки канал буде закрито, сервер відразу завершить свою роботу.

Щоб зупинити виконання сервера можна скористатись цією командою:

#service down sig

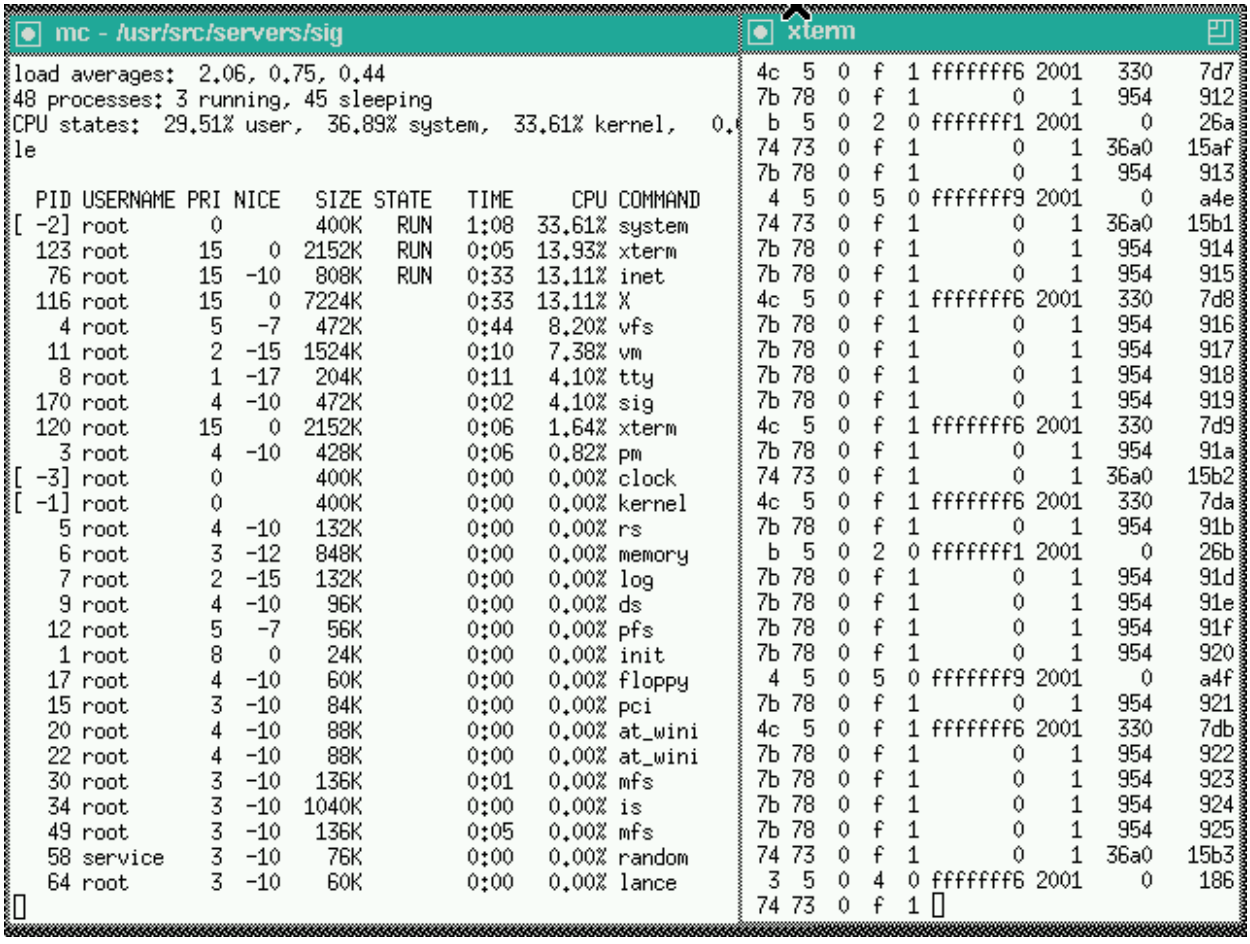


Рисунок 4.6 - Приклад роботи сервера сигнатур

На рисунку 4.6 показаний приклад виведення сигнатур процесів у реальному часі (праве вікно), та стан системи у вигляді таблиці запущених на виконання процесів (ліве вікно).

4.2 Програмні засоби дослідження нечітких експертних систем

Проведення дослідження було розділено на етап створення нечіткої експертної системи, вивчення її властивостей та етап використання експертної системи для вирішення задачі прогнозування стану процесів в персональному комп'ютері.

Серед існуючих програмних засобів моделювання нечітких експертних систем, для їх реалізації та дослідження використовувався пакет моделювання Matlab 6.1. На початкових етапах розробки нечітких експертних систем було використано GUI інтерфейс пакету системи Matlab, зокрема інструментальний засіб Fuzzy Logic Toolbox. Цей графічний інтерфейс дозволяє виконувати створення, моделювання, а також імпорт та експорт даних, використовуючи тільки його інструментальні можливості.

На рис. 4.7, 4.8а, 4.8б, 4.8в представлено результати моделювання у системі MatLab 6.1.

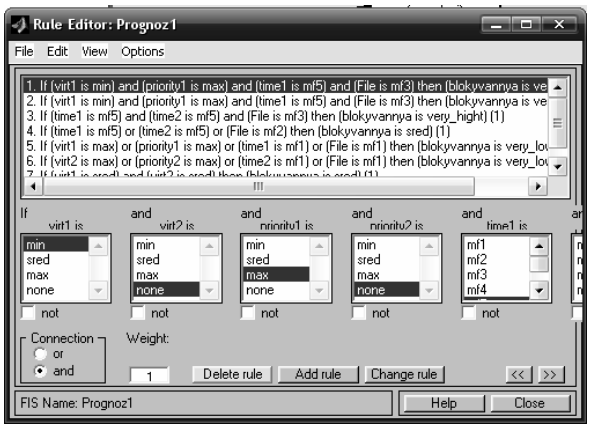
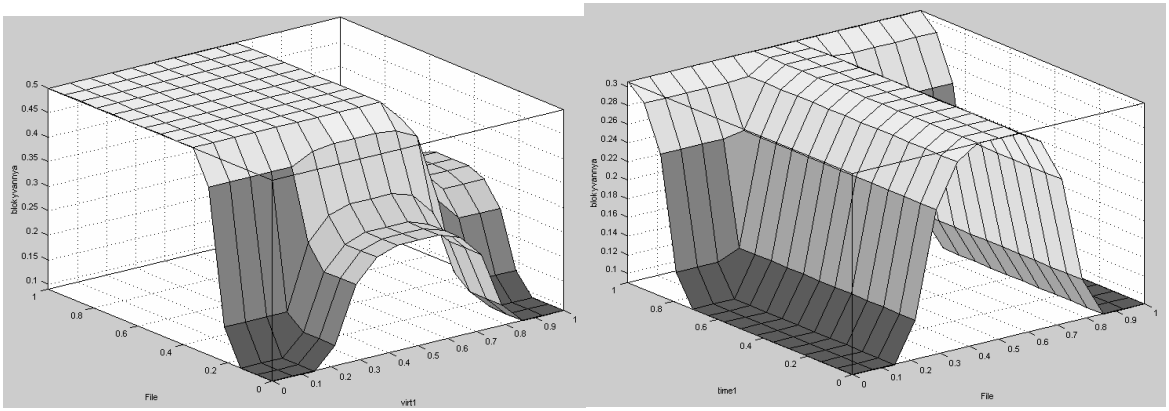
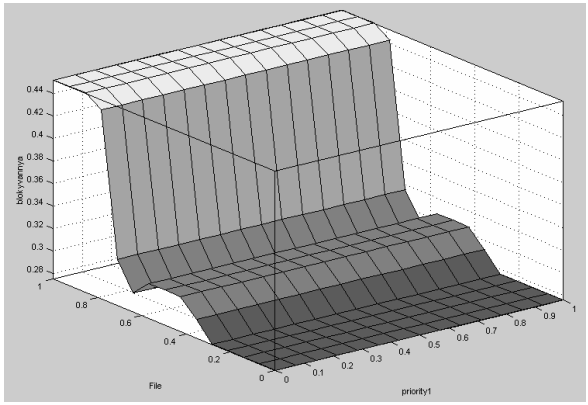


Рисунок 4.7 - Множина правил нечіткої підсистеми аналізу та логічного висновку



а) б)



в)

Рисунок 4.8 - Ймовірність настання взаємоблокування

4.3 Висновки

Розроблено програмні засоби для системи прогнозування взаємоблокувань процесів в КС.

Запропоновано алгоритм та програмні засоби для відстеження параметрів процесу на прикладі ОС сімейства Linux, на основі яких проводиться побудова сигнатури процесу.

Реалізовано підсистему логічного висновку з використанням нечіткої експертної системи.

ВИСНОВКИ

Проведено аналіз процесів та їх взаємодії у сучасних операційних системах. Виявлено, що при взаємодії процесів виникає проблема взаємного блокування їх роботи.

Проведено аналіз відомих методів вирішення задачі взаємоблокування процесів та виявлено їхні недоліки (блокування роботи ОС, наявність циклів активного очікування, складність програмної реалізації для багатьох процесів, необхідність використання спеціалізованої команди процесора).

Розроблено сигнатурну модель процесу, як об’єкту дослідження.

На основі сигнатурної моделі процесу розроблено метод та алгоритми прогнозування стану процесу в персональному комп’ютері.

На основі методу та алгоритмів прогнозування стану процесу реалізовано програмне забезпечення для сучасних операційних систем для підвищення безвідмовної роботи процесів, що виконуються на персональних комп’ютерах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Платонов Ю. М., Уткин Ю. Г. Диагностика зависания и неисправностей компьютера / Серия "Техномир". Ростов-на-Дону: "Феникс", 2001. – 320с.

2. Таненбаум Э, Вудхалл А. Операционные системы: разработка и реализация. Класика CS. – СПб: Питер, 2006. – 576с.

3. Дейтел Г. Введение в операционные системы: В 2-х т: Пер. с англ. Т. 1,2. – М: Мир, 1987. – 398с.

4. Вильям Столлингс. Операционные системы (4-е издание). – СПб: издательство "Вильямс", 2002. – 848с.

5. Таненбаум Э. Современные операционные системы: Перевод с английского. – СПб: Питер, 2002. – 1040с.

6. Дейтел Х.М., Дейтел П.Дж., Чофнес Д.Р. Операционные системы. Том 1: Основы и принципы: Перевод с английского. – М: "Бином", 2006. – 800с.

7. Хоар Ч. Взаимодействующие последовательные процессы: Пер. с англ. – М.: Мир, 1989. 264 с.

8. Элементы параллельного программирования /В.А. Вальковский, В.Е. Котов, А.Г. Марчук, Н.Н. Миренков; Под ред. В.Е. Котова. - М.: Радио и связь, 1983. – 240с.

9. Таненбаум Э., М.Ван Стеен. Распределенные системы. Принципы и парадигмы (Серия "Классика computer science"). – СПб: Питер, 2003. – 877с.

10. Робачевский А.М. Операционная система Unix (2 изд.). – СПб: BHV-Санкт-Петербург, 2007. – 656с.

11. Савенко О.С., Кльоц Ю.П., Мостовий С.В. Дослідження та аналіз блокування процесів в комп'ютерній системі // Вісник ХНУ – Хмельницький: ХНУ, 2007. – №3, т.1, с.248-251

12. Роберт Лав. Разработка ядра Linux (2-е издание). – СПб.: "Вильямс", 2006 г. – 448 с.

13. Стахнов А. Linux (2-е издание). – СПб.: "BHV-Санкт-Петербург", 2005 г. – 944с.

14. Савенко О.С., Мостовий С.В. Модель побудови сигнатури процесу в комп'ютерній системі // Прогресивні інформаційні технології в науці та освіті. Збірник наукових праць – Вінниця: Вінницький соціально-економічний інститут Університету "Україна", 2007. – с.58-63

15. Савенко О.С., Мостовий С.В. Порівняльний аналіз програмних засобів тестування системних ресурсів персональних комп’ютерів в локальній обчислювальній мережі // Материалы II научно-практической конференции "Образование и наука без границ-2006" – Днепропетровск: Наука и образование, 2006. – Том 8, с.115-118

16. Сигорский В.П. Математический аппарат инженера. – К.: издательство "Техніка", 1975. – 768с.

17. С.В. Мостовий. Модель сигнатури процесу в операційній системі // Вимірювальна та обчислювальна техніка в технологічних процесах – Хмельницький, 2007, №1. – С.66-68

18. Д.Рутковская, М.Пилиньский, Л.Рутковский. Нейронные сети, генетические алгоритмы и нечеткие системы. Перевод с польского И.Д.Рудинского. – М: Горячая линия – Телеком, 2006. – 452с.

19. В.В.Круглов, М.И.Дли, Р.Ю.Голунов. Нечеткая логика и искусственные нейронные сети. – М: Физматлит, 2001. – 224с.

20. Леоненков А.В. Нечеткое моделирование в среде MATLAB и fuzzyTECH. – СПб: БХВ-Петербург, 2005. – 736с.

21. Штовба С.Д. Проектирование нечетких систем средствами MATLAB. – М: Горячая линия – Телеком, 2007. – 288с.

22. E.G. Coffman, M,J, Elphick, A. Shoshani. System deadlocks // Computing Surveys, Vol.3, No.2, June 1971. – Pages: 67 – 78.

23. Nima Kaveh, Wolfgang Emmerich. Deadlock detection in distribution object systems // Software Engineering Notes, Vol.26, No.5, September 2001. – Pages 44 – 51.

24. Saddek Bensalem, Jean-Claude Fernandez, Klaus Havelund, Laurent Mounier. Confirmation of deadlock potentials detected by runtime analysis // International Symposium on Software Testing and Analysis – Portland, Maine, USA, 2006. – Pages: 41 – 50.

25. Савенко О.С., Мостовий С.В. Модель прогнозування стану процесів в комп'ютерній системі // Радіоелектронні і комп'ютерні системи – Харків: ХАІ, 2008. - №5 (32). – С.109-115

26. Мостовий С.В. Система прогнозування стану процесів в персональному комп'ютері // Сборник трудов VIII международной конференции "Интеллектуальный анализ информации" (ИАИ-2008) – Киев: Просвіта, 2008. – С.308-314

27. Мостовий С.В. Організація логічного висновку в системі прогнозування стану процесів в персональних комп'ютерах // Збірник праць X міжнародної науково-технічної конференції "Системний аналіз та інформаційні технології" (САІТ-2008) – Київ: Національний технічний університет України „Київський політехнічний інститут", 2008. – С.386

28. С.В. Мостовий. Представлення вхідних даних для системи прогнозування стану процесів в персональному комп'ютері // Вісник ХНУ – Хмельницький: ХНУ, 2008, №4 – С.145-148

29. Джеффри Рихтер. Windows для профессионалов. Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – СПб: Питер, 2003. – 752с.

30. Microsoft Develop Network http://msdn.microsoft.com/
    a. MSDN Library\Win32 and COM Development\Diagnostics\Performance Monitoring
    b. MSDN Library\Win32 and COM Development\Administration and Management\ Windows Management Instrumentation

c. MSDN Library\Win32 and COM Development\System Services\ DLLs, Processes, and Threads\ Processes and Threads

31. Бьерн Страуструп. Язык программирования C++. – М: "Бином", 2001. – 1099с.

32. Бовет Д., Чезати М. Ядро Линукс. – СПб: "БХВ-Петербург", 2007. – 1104с.

33. Дэвид Аллен. Переход с Windows на Linux. – СПб: "БХВ-Петербург", 2005. – 478с.

34. Даниэл Дж. Баррет. Основные команды Linux Linux: Основные команды. Карманный справочник. – М: "КУДИЦ-Образ", 2004. – 288с.

35. В. Г. Олифер, Н. А. Олифер. Сетевые операционные системы. – СПб: "Питер", 2002. – 544с.

36. Таненбаум Э. Операционные системы: разработка и реализация. – СПб: Питер, 2007. – 704с.

37. Christopher Negus. Linux Bible 2006 edition. Wiley, John & Sons, Incorporated. 2006. – 912p.

38. Linux for Programmers and Users. By Graham Glass, King Ables. – Prentice Hall, 2006. – 656p.

39. Pulse: A Dynamic Deadlock Detection Mechanism Using Speculative Execution. Tong Li1, Carla S. Ellis1, Alvin R. Lebeck1, and Daniel J. Sorin2 Department of Computer Science, Department of Electrical and Computer Engineering, Duke University. Proceedings of the 2005 USENIX Annual Technical Conference, Anaheim, California, April 10–15, 2005

40. The Deadlock Problem: An Overview Sreekaanth S. Isloor. T. Anthony Marsland, University of Alberta. Yair Amir, Home Page: http://www.cs.jhu.edu/~yairamir/cs418/600-418.html

41. Operating Systems (600.418), Lecture 4: Deadlocks

42. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the

Art of Virtualization. In Proceedings of the 20th ACM Symposium on Operating System Principles, pages 164-177, October 2003.

43. Левин Р., Дранг Д., Эделсон Б. Практическое введение в технологию искусственного интеллекта и экспертных систем с иллюстрациями на Бейсике. - М.: Финансы и статистика, 1990. – 112 с.

44. Питер Д. Введение в экспертные системы.: Пер. с англ.: Уч. пос. — М.: Издательский дом "Вильяме", 2001. — 624 с.

45. Aikins J. S. Prototypical knowledge for expert systems. Artificial Intelligence. №10. - 1983. - p. 163-210.

46. В.Н. Бондарев, Ф.Г. Аде Искусственный интеллект: Учеб. пособие для вузов. – Севастополь: Изд-во СевНТУ, 2002. – 615с.: ил.

47. Попов Э. В., Общение с ЭВМ на естественном языке. М.: Наука. 1982. – 187 с.

48. Giarratano J. and Riley G. Expert Systems: Principles and Programming, 2nd edn. Boston, MA: PWS Publishing. 1994. – p. 447-461.

49. Pega M., Sticklen J., Bond W., Functional Representation and Reasoning // IEEE Expert. April. 1993. - p. 65-78.

50. Forgy C. L. Rete: a fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence. №19. – 1982. - p. 17-37.

51. Ярушкина Н.Г. Методы нечетких экспертных систем в интеллектуальных САПР. - Саратов:Изд-во Сарат. ун-та, 1997. - 107 с.

52. Brachman R. J. and Schmoize J. G. An overview of the KL-ONE knowledge representation system. Cognitive Science, 9, 1985. - p. 171-216.

53. Goldberg A. and Robson D. Smalltalk-80: The Language and its Implementation. Reading, MA: Addison-Wesley. 1983. - p. 74-88.

54. Giarratano J. and Riley G. Expert Systems: Principles and Programming, 2nd edn. Boston, MA: PWS Publishing, 1994. – p. 447-461.

55. Shaw M. L. G., PLANET: some experience in creating an integrated system for repertory grid application on a microcomputers //

International Journal of Man-Machine Studies. Vol. 17, No. 3. 1982. - p. 345-360.

56. Richer M., An evaluating of expert system development tools // Expert Systems. 1986. - p. 166-183.

57. Тоценко В.Г. Обобщенная концепция экспертных систем диагностирования // Электронное моделирование. - 1995. – №5. - с.26-33.

58. Поморова О.В. Експертні системи як засіб підвищення ефективності діагностування цифрових та мікропроцесорних пристроїв: Дис. канд. техн. наук: 05.13.06. – Хмельницький, 2001. – 157 с.

59. Ивахненко А.Г., Юрачковский Ю.П. Моделирование сложных систем по экспериментальным данным. - М.: Радио и связь, 1987.- 118 с.

60. Нейронные сети в системах автоматизации / В.И. Архангельский, И.Н. Богаенко, Г.Г. Грабовский, Н.А. Рюмшин. – К.: Техніка, 1999. – 364 с.

61. Системы фуцци-управления / В.И. Архангельский, И.Н. Богаенко, Г.Г. Грабовский, Н.А. Рюмшин. – К.: Техніка, 1997. – 208 с.

62. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика. – М.: Горячая линия – Телеком, 2001. – 382 с.: ил.

63. Баршдорф Д. Нейронные сети и нечеткая логика. Новые концепции для технической диагностики неисправностей // Приборы и системы управления. - 1996. - №2. - с. 48-52.

64. Локазюк В.М., Поморова О.В. Принципи побудови та структура засобів фуцці-контролю і діагностування обчислювальних пристроїв та систем // Вісник Технологічного університету Поділля. - 2000. - №6. - с.139-141.

65. Локазюк В.М. Проблеми та методологія контролю та діагностування сучасних мікропроцесорних пристроїв та систем // Вимірювальна та обчислювальна техніка в технологічних процесах. - 2000. - №2. - С.10-17.

66. Леоненков А.В. Нечеткое моделирование в среде MATLAB и fuzzyTech. – СПб.: БХВ – Петербург, 2003. – 736 с.

67. Дюбуа Д., Прад А. Теория возможностей. Приложения к представлению знаний в информатике: Пер. с фр. – М.: Радио и связь, 1990. – 288 с.: ил.

68. Герасимов Б.В., Грабовский Г.Г., Рюмшин Н.А. Нечеткие множества в задачах проектирования, управления и обработки информации. – К.: Техніка, 2002. – 140 с.

69. Ротштейн А.П. Медицинская диагностика на нечеткой логике.- Винница:Континент – ПРИМ, 1996. – 132 с.

70. Marina H. Magalhaes, Rosangela Ballini, Rodrigo Goncalves Predictive Fuzzy Clustering Model for Natural Streamflow Forecasting. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 6 p.

71. Daniela Andone, Andrei Hossu Predictive Control Based on Fuzzy Model for Steam Generator. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 6 p.

72. Hamid Berenji, Yan Wang, David Vengerov, Reza Langari, Mo Jamshidi Using Gated Experts in Fault Diagnosis and Prognosis. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 5 p.

73. Медведев В.С., Потемкин В.Г. Нейронные сети. MATLAB 6. Под общ. Ред. к.т.н. В.Г. Потемкина. – М.: ДИАЛОГ-МИФИ, 2002. – 496 с.

74. Мишулина О.А., Лабинская А.А., Щербина М.В. Лабораторный практикум по курсу "Введение в теорию нейронных сетей". М.:МИФИ, 2000. – 204 с.

75. Омату С, Халид М, Юсоф Р. Нейроуправление и его приложения. М.: ИПРЖРБ, 2000. – 272 с.

76. Уоссермен Ф. Нейркомпьютерная техника. М.:Мир, 1992. – 453 с.

77. Чорненький В. І., Локазюк В. М. Особливості побудови генераторів тест-векторів та ідентифікаторів несправностей на основі штучних нейронних мереж // Вісник Технологічного університету Поділля. – 2000. - №6. – С.142-144.

78. О.В. Поморова Моніторинг та прогнозування станів комп‘ютерних пристроїв з використанням карти Кохонена // Тези доповідей VIII міжнародної конференції "Контроль і управління в складних системах". – УНІВЕРСУМ-Вінниця. – 2005. – С.113.

79. Быков А.П., Вейц А.В. От нейрона – к искусственному мозгу.-М.:Наука,1971.-128с.

80. Гельднер К. Кибернетика и ее будущее: Пер. с нем.-М.:Радио и связь, 1983. - 96 с.

81. Головко В.А. Нейроинтеллект: теория и применение. Книга 1. Организация и обучение нейронных сетей с прямыми и обратными связями. – Брест: Изд. БПИ, 1999. – 264 с.

82. Нейрокомпьютеры интеллектуальные роботы / Под. ред. Н.М.Амосова. –К.:Наук. думка, 1991. - 269 с.

83. Бовбель Е.И., Паршин В.В. Нейронные сети в системах автоматического распознавания речи // Зарубежная радиоэлектроника.-1998. - №4 .- с.49-65.

84. Поспелов Д.А., Пушкин В.Н. Мышление и автоматы. - М.:Сов.радио, 1972. – 224 с.

85. McCulloch W.S. and Pitts W. A Logical Calculus of Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophisics, 1943, Vol.5. – p. 574-589.

86. Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере. – Новосибирск, Наука, 1996. – 247 с.

87. Hebb D.O. The Organization of the Behavior // A Neuropsychological Theory.-N.-Y. John Wiley.-1949. - 409p.

88. Минский М., Пейперт С. Персептроны / Пер. с англ.-М.:Мир, 1971.-264 с.

89. Ивахненко А.Г. Самообучающиеся системы распознавания и автоматического управления. – К.:Техніка, 1969. - 392с.

90. Ивахненко А.Г., Мюлер Й.А. Самоорганизация прогнозирующих моделей. - К.:Техніка, 1985. - 223с.

91. Кохонен Т. Ассоциативные запоминающие устройства: Пер. с англ. – М.:Мир, 1982. – 384 с.

92. Логовский А.С., Якушев Д.Ж. Нейропакеты: что, где, зачем // Зарубежная радиоэлектроника. - 1997. - №2. - с.11-18.

93. Жаров А. Железо IBM. –М.:МИКРОАРТ, 1995. - 198с.

94. Глушков В.М. Основы безбумажной информатики. - М.:Наука, 1987. – 552 с.

95. Резник А.М. Нейрокомпьютеры. Часть 1: после нейрокомпьютерного бума // Компьютеры+Программы. - 1998. - №1. - с.8-13.

96. Степанов М.В. Оптические нейрокомпьютеры: современное состояние и перспективы // Зарубежная радиоэлектроника. - 1997. - №2. - с.32-56.

97. Резник А.М. Нейрокомпьютеры: подождем еще лет тридцать // Компьтеры + Программы. - 1998. - №4. - С. 48-50.

98. Дунин-Барковский В.Л. Информационные процессы в нейронных структурах. – М.: Наука, 1978. – 163 с.

99. Ефимов Е.И. Проблемы перебора в искусственном интеллекте // Техническая кибернетика, 1988, №2, с. 9-11.

100. Кохонен Т. Ассоциативная память. М.: Мир, 1982. – 239 с.

101. Кохонен Т. Ассоциативные запоминающие устройства. М.:Мир, 1982. – 384 с.

102. Dmitry V. Zhora Financial Forecasting using Random Subspace Classifier. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 6 p.

103. Hailin Li, Cihan H. Dagli, David Enke Forecasting Series-based Stock Price Data using Direct Reinforcement Learning. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 6 p.

104. Carmen Fierascu, Claudia Lidia Badea Counterpropagation with delays for Financial Prediction. Hailin Li, Cihan H. Dagli, David Enke Forecasting Series-based Stock Price Data using Direct Reinforcement Learning. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 5 p.

105. Georgina Stegmayer Volterra series Neural Networks to model an electronic device nonlinear behavior. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 4 p.

106. Antony Lam, Amar Raheja, Muthu Govindary Neural Network Models for Fabric Drape Prediction. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 5 p.

107. Marcelo Andrade Teixeira, Gerson Zaverucha Fuzzy Hidden Markov Predictor in Electric Load Forecasting. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 6 p.

108. N. Kussul, S. Skakun Neural Network Approach for User Activity Monitoring in Computer Networks. IJCNN2004 CD-ROM Conference Proceedings IEEE. Catalog Number: 04CH37541C. – 2004. – 5 p.

109. Дубровин В.И., Субботин С.А. Индивидуальное прогнозирование надежности изделий электронной техники на основе нейронных сетей // Труды VII Всероссийской конференции “Нейрокомпьютеры и их применение” - М.: ИПУ РАН, 2001. - С. 228-231.

Оцінка ефективності методу прогнозування взаємоблокувань процесів в комп'ютерній системі

Андрощук О.С., Горчилов І.І., Нагребецький О.В.
Хмельницький національний університет

При експлуатації комп'ютерних систем (КС) досить часто виникає ситуація блокування процесів, що виконуються у них. Частковим випадком блокування задач є можливість їх взаємного блокування [1]. Взаємне блокування – ситуація в багатозадачному середовищі або системах керування базами даних (СКБД), при якій кілька процесів перебувають у стані нескінченного очікування ресурсів, зайнятих самими цими процесами. Виникнення взаємних блокувань задач призводить до збільшення часу їхнього виконання (може зростати до нескінченості), до не ефективного використання ресурсів КС (порожні цикли очікування).

Тому однією із проблем виконання процесів в КС є уникнення входження їх в стан взаємоблокування. Дані питання досліджувались Дейкстрою, Хоаром, Брінч-Хансеном, Деккером, Петерсеном, Коффманом, Хольтом [2]. На сьогодні розроблена велика кількість методів та алгоритмів для уникнення взаємоблокувань процесів. Проте частина з них носить лише теоретичний характер, оскільки не може бути реалізованою в сучасних КС. Інша частина при реалізації стає достатньо громіздкою і ресурсоємною. Тому розробники сучасних операційних систем (ОС) сімейств Windows та Linux, а також розробники сучасних СКБД не включають відомі алгоритми уникнення взаємоблокувань процесів.

За умови завантаження системи процесами до 20% відсутність таких засобів була допустима. Проте стрімкий розвиток апаратних засобів, зростання об’єму та складності ПЗ, яке займається вирішенням масштабних та відповідальних задач, не повинно дозволяти виникнення взаємоблокування, що в свою чергу вимагає розробки нових підходів до вирішення цієї задачі.

Для усунення суттєвих недоліків відомих методів та алгоритмів вирішення проблеми взаємоблокування було розроблено метод прогнозування взаємоблокування процесів, що враховує життєвий шлях процесу [3].

Постановка задачі. З метою визначення доцільності використання та порівняння запропонованого методу з існуючими засобами вирішення задачі взаємоблокування необхідно провести оцінку його часової складності та ефективності.

Оцінка часової складності методу. Для оцінки часової складності методу прогнозування взаємоблокувань процесів було використано web-сервер Apache2 з PHP 5 та sql-сервер MySQL. Для перевірки роботи методу на сервері запускався на виконання PHP-скрипт, представлений в лістинзі 1,

ДОДАТОК А
(обов’язковий)
Копії публікацій

який при паралельному виконанні призводить до виникнення взаємоблокувань.

Лістинг 1

```
$sql = "SELECT COUNT(*) FROM t "; $x = mysql_query($sql);
$r = mysql_fetch_row($x);$max_id = $r[0];
$id1 = rand(0,$max_id); do { $id2 = rand(0,$max_id); } while ($id1==$id2);
mysql_query("START TRANSACTION;");
$sql1 = "select a from t where id = $id1 for update"; mysql_query($sql1);
//Вибір кортежу для обрахунку
usleep(100); //моделювання обробки даних
$sql2 = "select b from t where id = $id2 for update"; mysql_query($sql2);
mysql_query("COMMIT;"))
```

В представленому коді видалено бізнес-логіку, зате повністю збережено послідовність запитів, що при паралельному виконанні призводять до появи взаємних блокувань.

Рівні взаємоблокувань, представлені на рис. 1, показують взаємоблокування в КС. Суцільною лінією показано рівень взаємоблокувань при використанні стандартних засобів MySQL для виявлення заблокованих транзакцій. Пунктирною лінією показано рівень взаємоблокувань, що виникали при використанні запропонованого методу прогнозування взаємоблокувань.

Час виконання процесів при використанні різних механізмів вирішення взаємоблокувань показаний на рис. 2. Суцільною лінією показано середній час виконання процесів при використання стандартного механізму вирішення взаємоблокувань. Пунктирною – середній час виконання процесів при використанні запропонованого методу прогнозування взаємоблокувань. Штрих-пунктирною – середній час виконання процесів, за умови ненастання взаємоблокувань.

У випадку використання стандартного механізму виявлення взаємоблокувань спостерігається значне зростання часу виконання навіть при незначній кількості паралельних процесів. В цей самий період не спостерігається значного завантаження процесора (не більше 20%), оскільки процеси більшу частину часу очікують доступу до заблокованих ресурсів.

У випадку, коли не виникає взаємоблокувань процесів, час їх виконання зростає незначно (на 3% при кількості паралельних процесів 25 і рівні завантаження процесора не більше 20%).

Використання запропонованого методу прогнозування взаємоблокувань показує помітно пологіше наростання середнього часу виконання процесів, за таких самих умов.

Зниження часу виконання процесів при виконанні єдиного потоку

пояснюється відсутністю затримок, пов’язаних з очікуванням звільнення ресурсу.

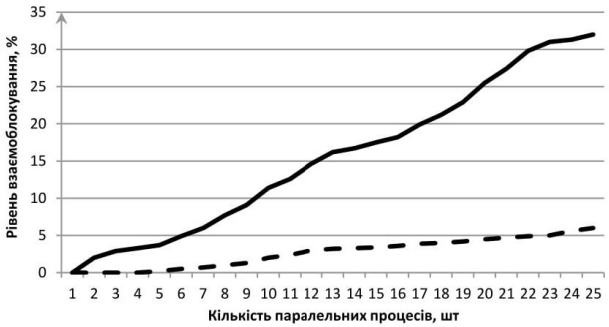


Рисунок 1 – Залежність рівня взаємоблокувань від кількості паралельних процесів

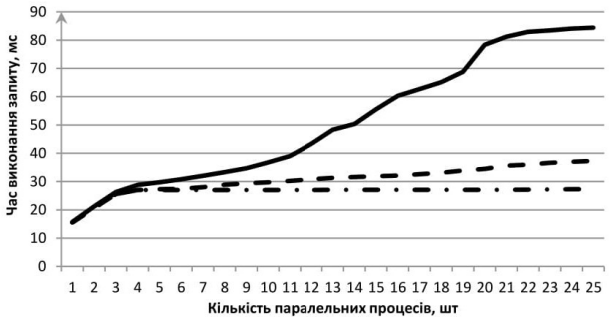


Рисунок 2 – Залежність часу обробки запитів від кількості паралельних процесів

В ході дослідження було проведено порівняння часу виконання фаз транзакцій при максимальному завантаженні системи і різних режимах роботи, що представлено на рис. 3.

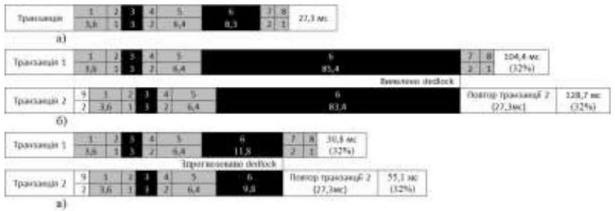


Рисунок 3 – Часові діаграми виконання фаз транзакції:

а) без взаємоблокування; б) взаємоблокування із стандартним виявленням; в) взаємоблокування із прогнозуванням. (1 – Підготовчі дії; 2 – Запуск транзакції; 3 – Очікування звільнення кортежу id1; 4 – Виконання 1-го SELECT; 5 – Опрацювання даних; 6 – Очікування звільнення кортежу id2; 7 – Виконання 2-го SELECT; 8 – Завершення транзакції; 9 – Затримка від початку запуску.)

На рис. 3.а представлена часова діаграма виконання фаз транзакції за умови, що при її виконанні не відбулось взаємоблокування. Середній час виконання транзакції склав 27,3мс, з них 11,3мс (41%) зайняло очікування заблокованих ресурсів. В ході дослідження таких транзакцій виявилось 36%.

На рис. 3.б представлено часову діаграму взаємоблокування двох транзакцій із стандартним механізмом їх вирішення. Транзакція 1 при першому звертанні (4) до таблиці БД вибирає та блокує кортеж з ключем id1. Транзакція 2 розпочинає виконуватись на 2мс пізніше, ніж Транзакція 1 і при першому звертанні (4) вибирає та блокує запис із ключем id2. Після опрацювання отриманих даних (5) транзакція 1 звертається до кортежу з id2. Оскільки він заблокований, то транзакція переходить до очікування звільнення ресурсу (6). В свою чергу транзакція 2 звертається до кортежу з id1. Він також виявляється заблокованим і транзакція 2 переходить в режим очікування звільнення ресурсу (6). Процеси потрапляють в ситуацію взаємного блокування і самостійно не можуть вийти з циклу нескінченного очікування. Взаємоблокування вирішує КС періодично аналізуючи граф процесів і ресурсів. Оскільки задача є алгоритмічно складною, то вона виконується через інтервали часу в 150-200мс. Після виявлення взаємоблокування один із процесів, що пізніше надійшов у систему, примусово завершується і повторно запускається на виконання. Інший з заблокованих процесів продовжує своє виконання. Великі часові проміжки між повторними аналізами графа призводять до значних затримок при виявленні взаємоблокувань. Середній час процесу, який був виконаний повторно складає 128,7мс, процесу, який продовжив роботу після взаємного блокування – 104,4мс. В ході дослідження таких транзакцій виявилось по

32%, оскільки процеси у взаємоблокування потрапляють парами.

На рис. 3.в представлено часову діаграму взаємоблокування двох транзакцій із прогнозуванням взаємоблокувань. До моменту запиту в транзакції 2 кортежу з ключем id2 часові діаграми ідентичні. В момент цього запиту (6) транзакція 2 потрапляє в граничний стан і для неї проводиться прогнозування. Час прогнозування складає близько 10мс. В результаті прогнозу визначається ймовірність взаємоблокування процесів, після чого обирається один із них. Цей процес знімається з виконання і запускається повторно. Середній час виконання процесу, що був виконаний повторно, складає 55,1мс; процесу, що продовжив роботу після взаємного блокування – 30,8мс. В ході дослідження таких транзакцій виявилось по 32%.

З отриманих досліджень середній час виконання транзакцій із стандартним механізмом виявлення взаємоблокувань склав: $27,3 \cdot 0,36 + 104,4 \cdot 0,32 + 128,7 \cdot 0,32 = 84,4$ мс, із прогнозуванням взаємоблокувань: $27,3 \cdot 0,36 + 30,8 \cdot 0,32 + 55,1 \cdot 0,32 = 37,3$ мс.

Застосування методу прогнозування взаємоблокувань забезпечило зменшення часу виконання процесів в $\frac{84,4}{37,3} = 2,3$ рази. Це дозволяє більш ефективно використовувати ресурси КС.

В роботі запропоновано оцінку часової складності та ефективності методу прогнозування взаємоблокувань поцесів в КС. Проведено його експериментальне дослідження на прикладі модифікованої СКБД MySQL. Середній час виконання процесу зменшився з 84,4мс при використанні стандартних засобів MySQL до 37,3 мс при використанні запропонованого методу. Отримані результати вказують на зменшення часових витрат на виконання процесів, що в свою чергу дозволяє опрацьовувати в 2,3 рази більшу кількість даних в одиницю часу для задач, в яких часто виникають взаємоблокування (відбувається конкурентна боротьба за ресурси).

Перелік посилань

1. Kaveh N. Deadlock detection in distribution object systems / Nima Kaveh, Wolfgang Emmerich // Software Engineering Notes. – September 2001. – Vol.26, No.5. – Pages 44 – 51.

2. Coffman E.G. System deadlocks / E.G. Coffman, M,J, Elphick, A. Shoshani. // Computing Surveys. – June 1971. – Vol.3, No.2. – Pages 67 – 78.

3. О.С. Савенко, С.В. Мостовий. Прогнозування потрапляння процесів у стан взаємоблокування в комп'ютерних системах. - Труды XIII Международной научно-практической конференции "Современные информационные и электронные технологии" (СИЭТ 2012). - Одесса: Одесский национальный политехнический университет,2012.- ст. 74-75

ДОДАТОК Б

(обов’язковий)

Презентація роботи

1

Хмельницький національний університет
Факультет програмування та комп’ютерних і телекомунікаційних систем
Кафедра кібербезпеки та комп’ютерних систем і мереж

Горчилов Іван Іванович

Інформаційна технологія діагностування КС на наявність ваємоблокувань процесів

Спеціальність: 123 – Комп’ютерна інженерія

Науковий керівник: доктор технічних наук, професор Андрощук О.С.

ЗАГАЛЬНА ХАРАКТЕРИСТИКА МАГІСТЕРСЬКОЇ РОБОТИ

Метою роботи є розроблення методу, алгоритмів та відповідних їм програмних засобів прогнозування стану процесів, що виконуються в комп'ютерних системах, для підвищення надійності роботи КС

Об'єкт дослідження – процес прогнозування стану взаємоблокування процесів в комп'ютерних системах.

Предмет дослідження – методи та засоби прогнозування стану взаємоблокування процесів в комп'ютерних системах.

Задачі дослідження:

1) дослідити процеси у сучасних комп'ютерних системах; провести аналіз сучасних методів за засобів вирішення задачі взаємоблокування процесів, виявити недоліки в їх роботі, проаналізувати їх ефективність; дослідити можливість здійснення прогнозування стану взаємоблокування процесів в КС з використання компонентів штучного інтелекту;
2) розробити модель процесів, що виконуються в КС, та модель процесу прогнозування стану взаємоблокування процесів;
3) розробити метод прогнозування стану взаємоблокування процесів в КС, який дає можливість уникати взаємоблокування процесів та є простим у реалізації для сучасних КС;
4) розробити алгоритми прогнозування стану взаємоблокування процесів та дослідити їх складність;
5) реалізувати метод прогнозування стану взаємоблокування процесів у вигляді програмного забезпечення та впровадити їх у виробництво з метою підвищення надійності роботи КС.

Методи дослідження базуються на основних положеннях методів аналізу даних, нечіткої логіки, теорії графів, теорії множин.

Наукова новизна одержаних результатів:

1. модель процесів, що виконуються в КС, яка дозволяє однозначно ідентифікувати процес на ПК та визначити його стан
2. модель процесу прогнозування стану взаємоблокування процесів в КС, яка дозволяє виділити множину процесів, що наближаються до стану взаємоблокування, і визначити серед них ті, що наймовірніше потраплять у стан взаємоблокування.
3. метод прогнозування стану взаємоблокування процесів в КС, який розв'язує задачу взаємоблокування процесів і, на відміну від відомих методів, має однакову складність реалізації для різних КС, що дозволяє підвищити надійність роботи КС

Апробація роботи. Наукові результати і основні положення кваліфікаційної роботи магістра доповідались і обговорювались на всеукраїнській і міжнародній науково-практичних конференціях.

Публікації. За темою кваліфікаційної роботи опубліковано 1 стаття у збірнику наукових праць.

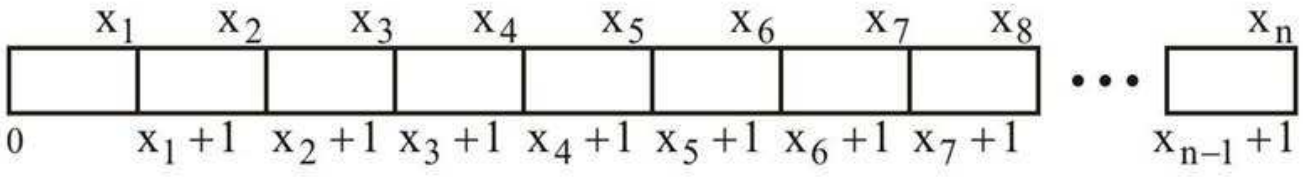
Класифікація методів вирішення задачі взаємоблокування процесів

- **ігнорування проблеми взаємоблокування** у випадку дуже низької ймовірності виникнення взаємоблокування;
- **методи, що запобігають взаємоблокуванням** шляхом створення передумов для невиконання однієї з чотирьох умов настання взаємоблокування;
- **методи виявлення взаємоблокувань**, які полягають у тому, щоб завжди задовільняти запити на ресурси, коли це можливо, і періодично перевіряти систему на наявність взаємоблокувань та вирішувати проблему у випадку її настання;
- **методи уникнення взаємоблокування**, що полягають у тому, щоб не задовільняти запит на ресурс, якщо його виділення може потенційно спричинити взаємоблокування.

Недоліки існуючих методів уникнення та виявлення взаємоблокувань:

№ п/п	Назва методу	До якого класу належить метод	Реалізація	Недоліки
1	Заборона на переривання	Запобігання взаємоблокуванню	Апаратна	Блокування роботи ОС, активне очікування
2	Змінні блокування	Уникнення взаємоблокування	Програмна	Часткове вирішення задачі блокування, активне очікування
3	Строге чередування	Уникнення взаємоблокування	Програмна	Зниження продуктивності роботи обчислювальної системи, активне очікування
4	Алгоритм Деккера	Уникнення взаємоблокування	Програмна	Складність програмної реалізації, активне очікування
5	Алгоритм Петерсона	Уникнення взаємоблокування	Програмна	Складність програмної реалізації для багатьох процесів, активне очікування
6	Семафори	Запобігання взаємоблокуванню	Програмна	Можливість блокування через порушення порядку використання, активне очікування
7	М'ютекси	Запобігання взаємоблокуванню	Програмно-апаратна	Наявність спеціалізованої команди процесора
8	Монітори	Запобігання взаємоблокуванню	Програмна	Необхідність підтримки на рівні компіляторів
9	Алгоритм банкіра	Уникнення взаємоблокування	Програмна	Складність програмної реалізації для багатьох процесів, прив'язка до ресурсів

Представлення сигнатури процесу



- **ідентифікатор процесу (x_1 біт);**
- **ідентифікатор батьківського процесу (x_2-x_1 біт);**
- **ідентифікатор користувача (x_3-x_2 біт);**
- **пріоритет процесу (x_4-x_3 біт);**
- **кількість дескрипторів файлів, що використовуються процесом (x_5-x_4 біт);**
- **обсяг віртуальної пам'яті, що використовує процес (x_6-x_5 біт);**
- **час виконання процесу (x_7-x_6 біт);**
- **додаткові параметри процесу (x_n-x_7 біт).**

Діаграма станів процесу та його життєвий цикл

Під сигнатурою процесу будемо розуміти сукупність його характеристик, яка однозначно ідентифікує його стан в КС в певний момент часу t:

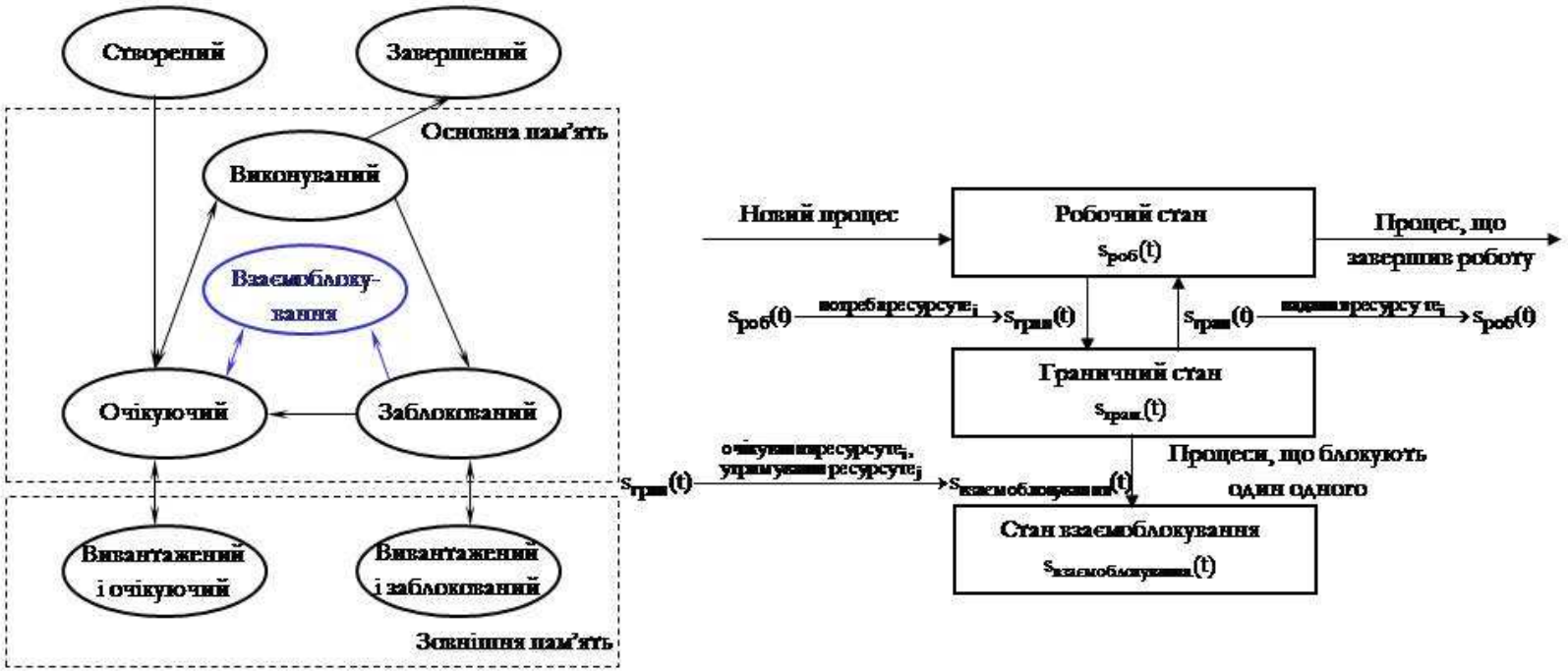
$$a_i(t) \rightarrow (a_{i_1}^t, a_{i_2}^t, ..., a_{i_c}^t)$$

Позначимо стан процесу через w, причину зміни параметрів через r, а перехід із стану в $W_i \xrightarrow{\text{зміна параметрів з причини } r_j} W_{i+1}$. Тоді життєвий цикл процесу буде мати вигляд:

$$a_i : W_0 \xrightarrow{r_1} W_1 \xrightarrow{r_2} ... \xrightarrow{r_{k-1}} W_{k-1} \xrightarrow{r_k} W_k$$

$$a_i : (a_{i_1}^{t_0}, a_{i_2}^{t_0}, ..., a_{i_c}^{t_0}) \xrightarrow{r_1} (a_{i_1}^{t_1}, a_{i_2}^{t_1}, ..., a_{i_c}^{t_1}) \xrightarrow{r_2} (a_{i_1}^{t_{k-1}}, a_{i_2}^{t_{k-1}}, ..., a_{i_c}^{t_{k-1}}) \xrightarrow{r_k} (a_{i_1}^{t_k}, a_{i_2}^{t_k}, ..., a_{i_c}^{t_k})$$

Життєвий цикл процесів з врахуванням стану взаємоблокування



Життєвий шлях процесу

1) процес створений, йому надано всі необхідні для завершення роботи ресурси, він виконується і завершується (процес весь час знаходиться в робочому стані $s_{роб}(t)$), тобто:

$$a_i : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k$$

2) процес створений, йому надано частину ресурсів, необхідних для завершення роботи, він потребує додаткових ресурсів, через деякий час отримує їх, виконується і завершується (процес спочатку знаходиться в робочому стані $s_{роб}(t)$, потім переходить $s_{роб}(t) \xrightarrow{\text{потребаресурсу } re_i} s_{гран}(t)$ в граничний стан $s_{гран}(t)$, потім $s_{гран}(t) \xrightarrow{\text{наданняресурсу } re_i} s_{роб}(t)$ знову в робочий стан $s_{роб}(t)$), тобто:

$$a_i : s_0 \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_l \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_k$$

3) процес створений, йому надано частину ресурсів, необхідних для завершення роботи, він потребує додаткових ресурсів, які утримує інший процес, який в свою чергу потребує ресурсів першого процесу (процес спочатку знаходиться в робочому стані $s_{роб}(t)$, потім переходить $s_{роб}(t) \xrightarrow{\text{потребаресурсу } re_i} s_{гран}(t)$ в граничний стан $s_{гран}(t)$, із якого відбувається перехід $s_{гран}(t) \xrightarrow[\text{утримуванняресурсу } re_j]{\text{очікуванняресурсу } re_i} s_{взаємоблокування}(t)$ до стану взаємоблокування).

$$\begin{cases} a_i : s_0 \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_l \xrightarrow{r_j} s_z \\ a_m : s_0 \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_n \xrightarrow{r_j} s_z \end{cases}$$

Стани комп'ютерної системи

Комп'ютерна система може знаходитись у наступних станах:

1) Робочий стан – всім процесам надано всі необхідні для завершення роботи ресурси (процеси весь час знаходиться в робочому стані $s_{роб}(t)$), тобто:

$$\begin{cases} a_0 : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k \\ \vdots \\ a_i : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k \end{cases}$$

2) Граничний стан – частині процесів необхідно для завершення роботи додаткових ресурсів, які вони отримують через деякий час (процеси спочатку знаходиться в робочому стані $s_{роб}(t)$, потім переходять $s_{роб}(t) \xrightarrow{\text{потреба ресурсу } r_j} s_{гран}(t)$ в граничний стан $s_{гран}(t)$, потім $s_{гран}(t) \xrightarrow{\text{надання ресурсу } r_j} s_{роб}(t)$ знову в робочий стан $s_{роб}(t)$), тобто:

$$\begin{cases} a_0 : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k \\ \vdots \\ a_n : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k \\ a_{n+1} : s_0 \xrightarrow{r_j} \dots \xrightarrow{r_j} s_l \xrightarrow{r_j} \dots \xrightarrow{r_j} s_k \\ \vdots \\ a_i : s_0 \xrightarrow{r_j} \dots \xrightarrow{r_j} s_l \xrightarrow{r_j} \dots \xrightarrow{r_j} s_k \end{cases}$$

3) Стан взаємоблокування процесів - процесам надано частину ресурсів, необхідних для завершення роботи, але вони потребують додаткових ресурсів, які утримують інші процеси, що в свою чергу потребують ресурсів, які надані першій групі процесів:

$$\begin{cases} a_0 : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k \\ \vdots \\ a_m : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k \\ a_{m+1} : s_0 \xrightarrow{r_j} \dots \xrightarrow{r_j} s_l \xrightarrow{r_j} s_x \\ \vdots \\ a_i : s_0 \xrightarrow{r_j} \dots \xrightarrow{r_j} s_n \xrightarrow{r_j} s_x \end{cases}$$

Подамо прогнозування стану процесів наступною моделлю:

$$M = \langle A, S, D, P, R \rangle$$

де A **– множина сигнатур процесів, що виконуються в КС у даний момент;**

S **– впорядкована послідовність характеристик комп'ютерної системи (загальний обсяг оперативної пам'яті, зовнішньої пам'яті, обсяг вільної в даний момент пам'яті, кількість периферійних пристроїв, кількість активних процесів користувача та ядра системи);**

D **– підмножина сигнатур процесів, що знаходяться у стані, наближеному до стану взаємоблокування;**

P **– множина правил, на основі яких проводиться розбиття множини сигнатур працюючих процесів на дві підмножини: підмножину сигнатур процесів, що знаходяться у стані, наближеному до стану взаємоблокування, та підмножину сигнатур процесів, які не досягли цього стану;**

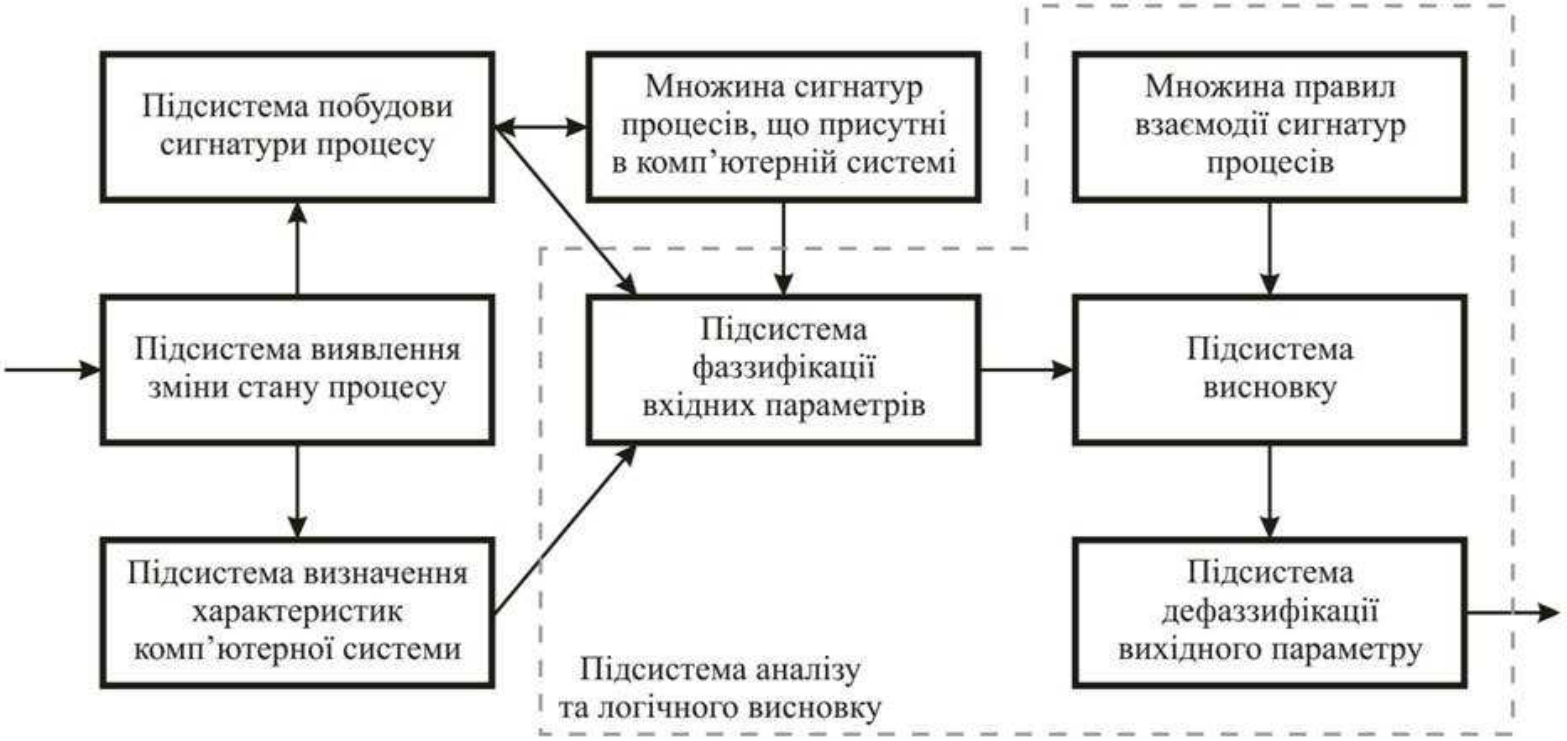
R **– вектор ймовірностей переходу у стан блокування процесів із підмножини** D.

Прогнозування стану процесів в персональному комп'ютері

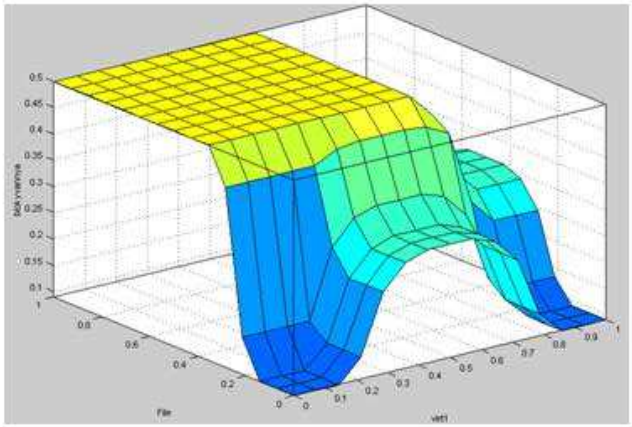
Метод прогнозування стану процесів в ПК включає наступні етапи:

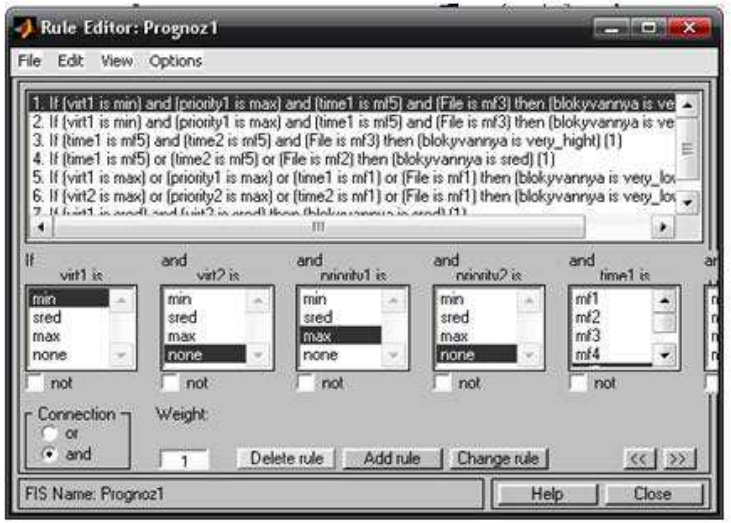
- Виявлення змін в системі
- Побудова сигнатури процесів
- Визначення основних характеристик ПК та ОС
- Приведення до нормованого вигляду вхідних параметрів та їх предача на підсистему аналізу та логічного висновку
- Здійснення висновку про настання взаємоблокування процесів

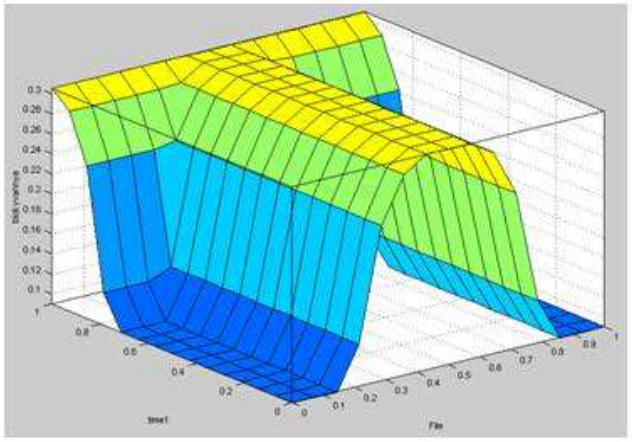
Система прогнозування стану процесів в персональних комп'ютерах

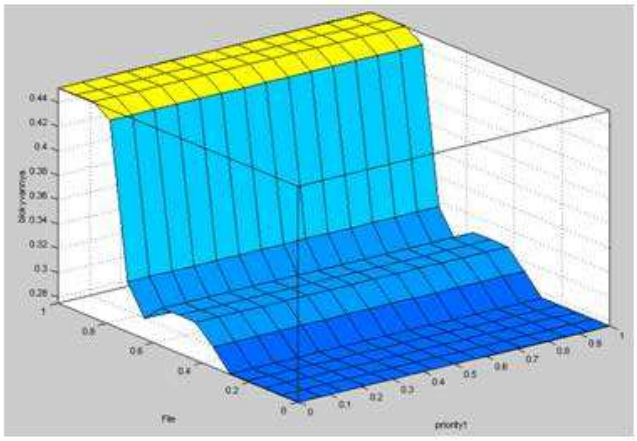


Підсистема аналізу та логічного висновку

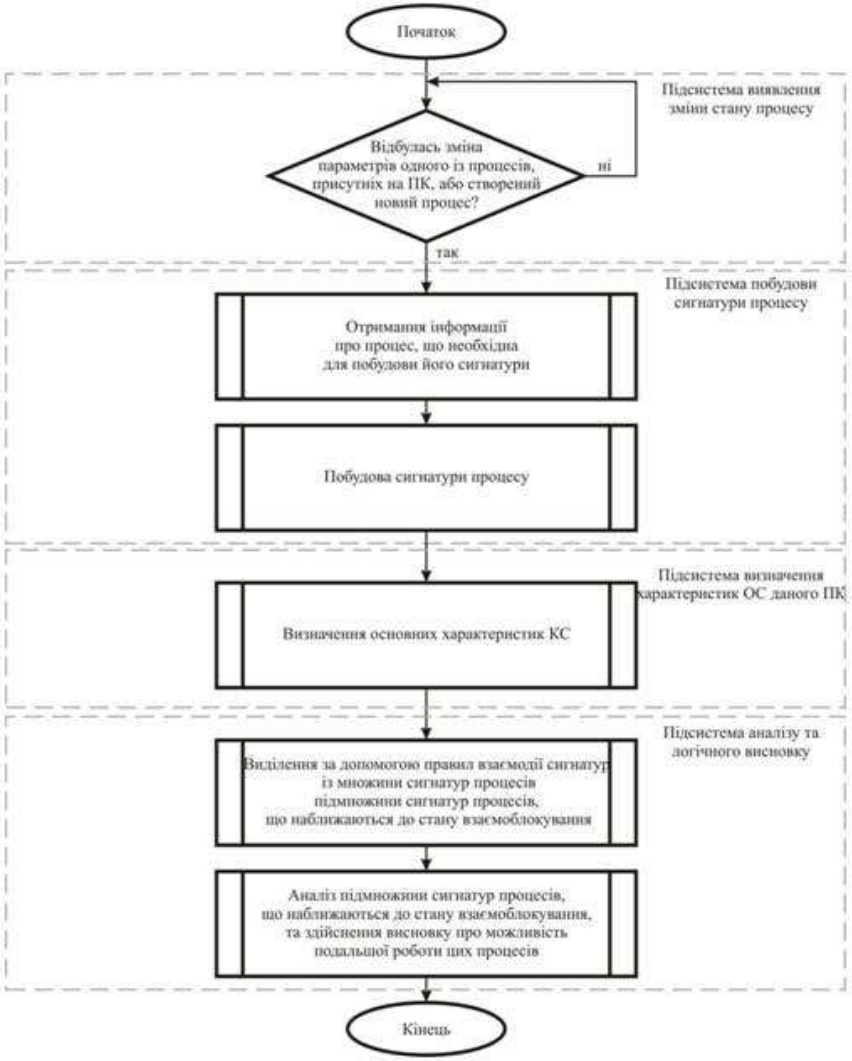




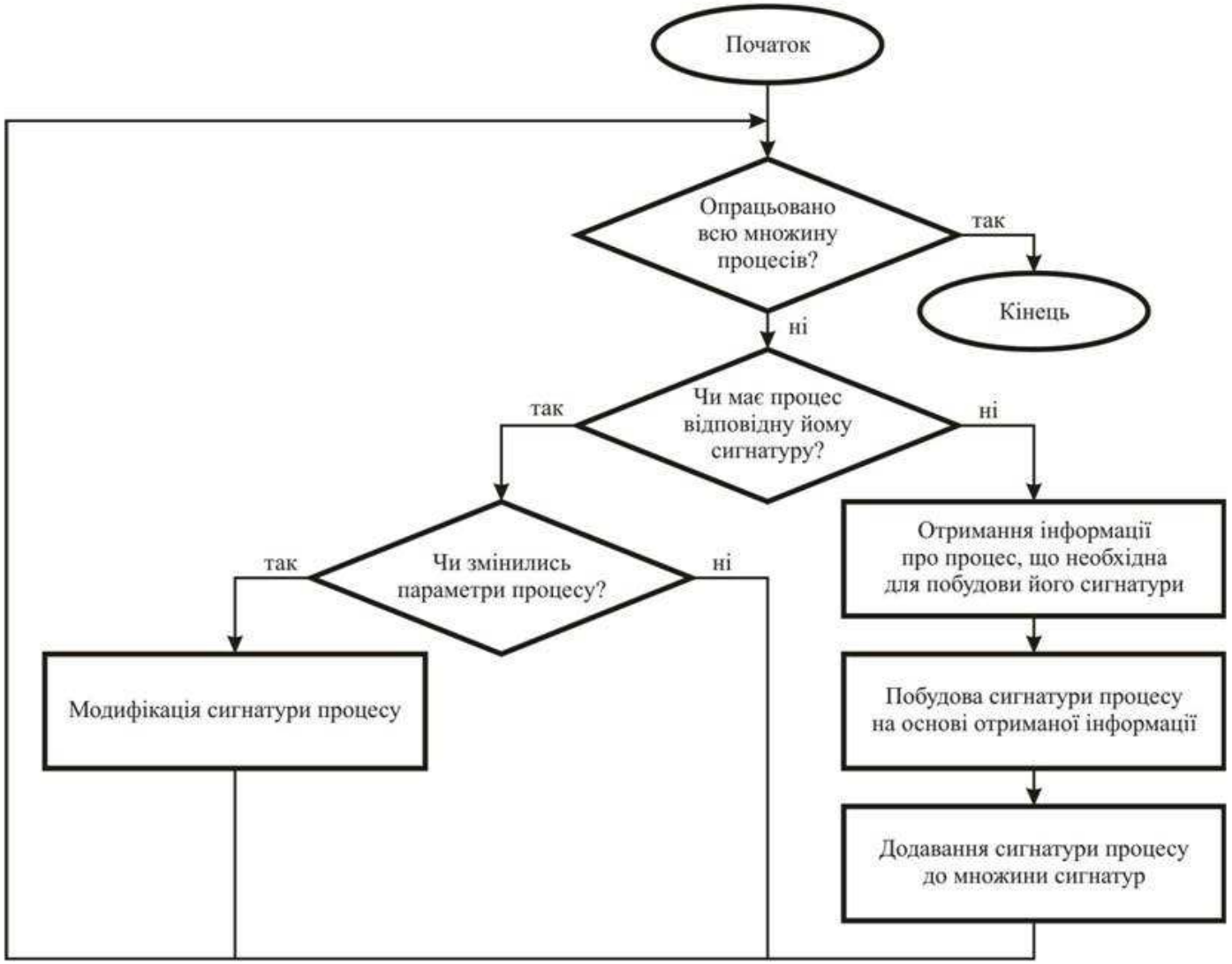




Алгоритм прогнозування стану процесу



Алгоритм побудови сигнатури



Виявлення процесів, що можуть потрапити у стан взаємоблокування (виявлення процесів, що знаходяться у граничному стані)

1. Якщо $a_k \in A$ потребує ресурс $r_i \in R$, то перехід до 2.
2. Якщо $a_k \in A$ знаходиться в стані $s_{роб}(t)$, то перехід до 3, інакше перехід до 4.
3. Виконати для $a_k \in A$ перехід із робочого стану до граничного $s_{роб}(t) \xrightarrow{\text{потреба ресурсу } r_i} s_{гран}(t)$ та включити $a_k \in A$ до $D \subset A$. Перехід до 4.
4. Якщо $a_k \in A$ надано у використання ресурс $r_i \in R$, то перехід до 5, інакше перехід до 6.
5. Виконати для $a_k \in A$ перехід із граничного стану до робочого $a_{гран}(t) \xrightarrow{\text{надання ресурсу } r_i} s_{роб}(t)$ та виключити $a_k \in A$ із $D \subset A$. Перехід до 6.
6. Якщо перевірено всю множину A, то перехід до 7, інакше перехід до 1.
7. Кінець алгоритму.

1. Проводимо нормування показників за наступною формулою:

$$P_n = 1 - \frac{P_d + P_m}{P_d \cdot P_m}, \; Pd \neq 0, \tag{1}$$

де P_n – черговий нормований показник;

P_d – поточне значення показника в конкретній системі;

P_m – максимальне значення показника в конкретній системі.

2. Для кожного $x_g \in X$, $X = D \cup S$, $g = \overline{1,h}$ визначаємо ступінь належності до нечітких множин входу (ступені істинності $\mu_g^j(x_g)$).

3. На основі ступенів істинності передумов $\mu_g^j(x_g)$ для кожного правила P_j, $j = \overline{1,n}$ розраховуємо ступінь його виконання α_j за формулою (2).

$$\alpha_j = \min\left(\mu_1^j(x_1), \mu_2^j(x_2), \ldots \mu_k^j(x_k)\right) \tag{2}$$

4. На основі ступеню виконання α_j для кожного правила P_j, $j = \overline{1,n}$ розраховуємо результат його виконання.

5. На основі результату виконання кожного правила P_j, $j = \overline{1,n}$ визначаємо вихідну нечітку множину з усіченою функцією приналежності $\overline{\mu}^j(y)$ за формулою (3).

$$\overline{\mu}^j(y) = \min\left(\alpha_j, \mu^j(y)\right) \tag{3}$$

6. Вихідні нечіткі множини $\overline{\mu}^j(y)$ згідно (4) агрегуємо в нечітку множину висновку $\tilde{y}$, що має функцію приналежності (5).

$$\tilde{y} = \max(\mu^j(y)), j = \overline{1,r} \quad (4)$$

$$\mu_{\tilde{y}} = \max(\mu^1(y), \mu^2(y), ..., \mu^r(y)) \quad (5)$$

7. Приводимо до чіткості нечітку множину $\tilde{y}$ за допомогою процедури дефазифікації центроїдним методом (6)

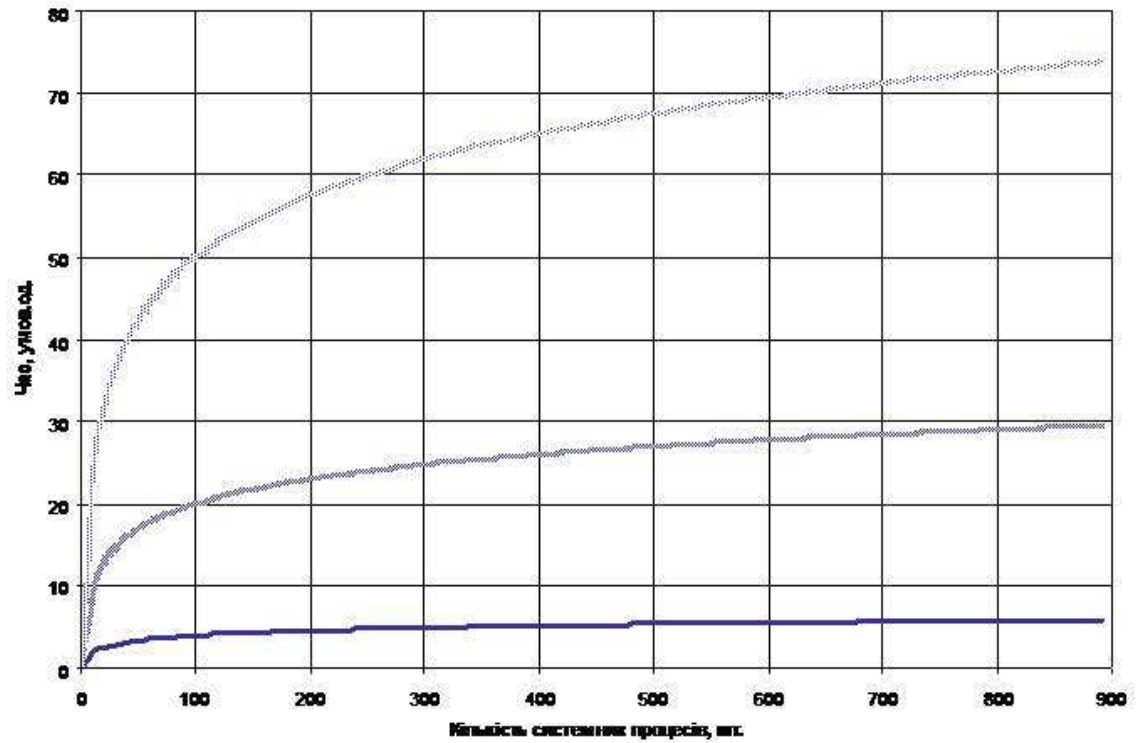
$$\bar{y} = \frac{\int_{c_1}^{c_2} x \cdot f_{\tilde{y}}(x)dx}{\int_{c_1}^{c_2} f_{\tilde{y}}(x)dx} \quad (6)$$

8. Встановлюємо для R_k значення $\bar{y}$.

9. Повторюємо кроки 1-8 k разів.

10. Кінець алгоритму.

Оцінка часової складності алгоритму прогнозування стану взаємоблокування процесів



Результати порівняння розробленого методу та відомих методів

№ п/п	Параметр для порівняння	Назва методу								
		Заборона на переривання	Строге чередування	Алгоритм Деккера	Алгоритм Петерсона	Алгоритм банкіра	Семафори	Монітори	М'ютекси	Прогнозування стану процесів
1	Складність реалізації для 2 і більше процесів	ні	ні	так	так	так	ні	ні	ні	ні
2	Прив'язка до кількості та видів ресурсів ПК	ні	ні	ні	ні	так	ні	ні	ні	ні
3	Активне очікування	так	так	так	так	так	так	ні	ні	так
4	Необхідність підтримки на рівні компіляторів	ні	ні	ні	ні	ні	ні	ні	так	ні
5	Наявність спеціалізованої команди процесора	ні	ні	ні	ні	ні	ні	так	ні	ні
6	Зниження продуктивності роботи обчислювальної системи	ні	так	ні	ні	ні	ні	ні	ні	ні
7	Блокування роботи операційної системи	так	ні	ні	ні	ні	так	ні	ні	ні
8	Спосіб реалізації методу	апаратний	програмний	програмний	програмний	програмний	програмний	програмний	програмно-апаратний	програмний

Висновки

1. Досліджено процеси у сучасних комп'ютерних системах; проведено аналіз сучасних методів за засобів вирішення задачі взаємоблокування процесів з виділенням недоліків, суть яких полягає у складності програмної реалізації для багатьох процесів, зниженні продуктивності КС, наявності циклів активного очікування, блокуванні роботи КС, необхідності наявності та використанні спеціалізованої команди процесора.
2. Розроблено модель процесів, що виконуються в КС, яка дозволяє однозначно ідентифікувати процес та визначити його стан, та модель процесу прогнозування стану взаємоблокування процесів, яка дозволяє виділити множину процесів, що наближаються до стану взаємоблокування, і визначити серед них ті, що наймовірніше потраплять у стан взаємоблокування.
3. Розроблено метод прогнозування стану взаємоблокування процесів в КС, який дає можливість уникати взаємоблокування процесів та є простим у реалізації для сучасних КС;
4. Реалізовано метод прогнозування стану взаємоблокування процесів у вигляді програмного забезпечення, яке дозволяє підвищити надійність роботи КС.

Tue Dec 08 07:34:57 EET 2020, Муляра І.В, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 15.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 10%**

ID: 82872 Название: Інформаційна технологія діагностування КС на наявність ваємоблокувань процесів Добавлено в БД: 2020-12-08 Авторы: Горчилов І.І. Руководители: Андрощук О.С. Консультанты: Опоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	117282	1028	21429 (18%)	195 (19%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы
511	Название: Програмне забезпечення підсистеми планування роботи процесів операційної системи "Minix" Добавлено в БД: 2010-06-10 Авторы: Бойко Ігор Ігорович Руководители: Кльоц Юрій Павлович Консультанты: Нестер Анатолій Антонович, Гарват Ольга Анатоліївна Опоненты:	17380 (15.0%)	148 (14.0%)

UNICHECK

User name:
Кафедра кибербезпеки

Check ID:
1005395923

Check date:
08.12.2020 08:28:01 EET

Check type:
Doc vs Internet

Report date:
08.12.2020 08:29:14 EET

User ID:
100005590

File name: **Горчилов_КРМ_плагіат**

Page count: **112** Word count: **18873** Character count: **142940** File size: **3.36 MB** File ID: **1005687938**

Text modifications detected (similarity score might be affected)

15.6% Matches

Highest match: **7.2%** with Internet source (https://journals.khnu.km.ua/vestnik/pdf/tech/2014_1/19.pdf)

15.6% Internet sources | 739 ..... Page 114

No Library search was conducted

0% Quotes

Exclusion of quotes is off

Exclusion of references is off

0% Exclusions

No exclusions

Modifind

Text modifications detected. Find more details in the online report.

Replaced characters | 30

Suspicious formatting | 20 Pages

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Інформаційна технологія діагностування КС на наявність ваємоблокувань процесів

Автор: Горчилов Іван Іванович

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма:Програмування та захист комп'ютерних систем і мереж

Науковий керівник:Андрошук О.С, д.т.н., професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих методів та технологій, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення;
4) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 15.6% і адресується до 739 першоджерел, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О.С. Андрошук

Завідувач кафедри КбКСМ, гарант ОП _____ Ю.П. Кльоц

Дата: 10.12.2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «магістр»

Магістр Горчилов Іван Іванович

Тема Інформаційна технологія діагностування КС на наявність ваємоблокувань процесів

Спеціальність 123 – Комп'ютерна інженерія

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «магістр»:

кількість листів креслень 23 ; кількість сторінок записки 141

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі розроблено метод прогнозування взаємоблокувань прцесів у КС для підвищення надійності функціонування КС .

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині роботи

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі подана загальна характеристика поставленої задачі, чітко визначено об'єкт, предмет та методи дослідження, сформульована актуальність. Визначені задачі, які необхідно вирішити для досягнення поставленої мети, практична цінність отриманих результатів, їхня новизна та наведені відомості про публікації. У першому розділі проведено огляд існуючих в комп'ютерних системах методів вирішення взаємоблокувань, виконане обгрунтування актуальності теми дослідження і виконана постановка задачі. В другому розділі виконана розробка моделі для реалізації методу, яка базується на основних положеннях методів аналізу даних, нечіткої логіки, теорії графів, теорії множин. В третьому розділі визначено основні положення методу та розроблено алгоритми його реалізації. Четвертий розділ присвячено апробації методу та алгоритмів його реалізації моделюванням.

4. Позитивні сторони роботи Кваліфікаційна робота має комплексну наукову і практичну цінність. Наукова цінність полягає у розробці методу прогнозування взаємоблокування процесів в комп'ютерних системах для підвищення надійності функціонування комп'ютерних систем. Практична цінність результатів дослідження полягає у обґрунтуванні можливості реалізації та застосування даного методу у сучасних КС під керуванням різноманітних операційних систем .

5. Негативні сторони роботи В роботі відсутній деталізований опис технічних засобів реалізації методу (програмних або програмно-апаратних).

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням стандартів. В загальному графічне оформлення виконане якісно, пояснювальна записка відповідає нормам щодо її оформлення.

7. Відгук про роботу в цілому В загальному кваліфікаційна робота заслуговує позитивної оцінки. Весь матеріал кваліфікаційної роботи структурований, чіткий та послідовний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики кваліфікаційної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу для досягнення поставленої мети.

8. Інші зауваження Окремі описи в пояснювальній записці подано стисло, що ускладнює сприйняття матеріалу наукової роботи фахівцями в обраній предметній галузі. Не на всі джерела з переліку є посилання в тексті пояснювальної записки

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Гурман Іван Васильович
кандидат технічних наук, доцент,
доцент кафедри інженерії програмного забезпечення

« 10 » грудня 2020. (підпис)

ХНУ. Зам. 54, тир. 6000, 2007.