Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: Розробка методу та програмних засобів для прогнозування ситуації взаємних блокувань

Виконав: студентII курсу
групи 1ПІ-18 м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Завертайло Костянтин Сергійович
(прізвище та ініціали)
Керівник:к.т.н., доц. каф. ПЗ
Хошаба Олександр Мирославович
(прізвище та ініціали)
Рецензент:к.т.н., доц. каф. КН
Богач Ілона Віталіївна
(прізвище та ініціали)

ВНТУ – 2019

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
"____"_____2019 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Завертайло Костянтину Сергійовичу

1. Тема роботи – розробка методів і програмних засобів для прогнозування ситуації взаємних блокувань.

Керівник роботи: Хошаба Олександр Мирославович, к.т.н., затверджені наказом вищого навчального закладу від "__"_____2019 року №___

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: методи розробки планувальників в операційних системах, методи розробки міжпроцесної взаємодії в операційних системах, методи розробки системних викликів в операційних системах, методи розробки ядра операційної системи, теорія множин та нечітких множин, методи приведення до фазифікації та дефазифікації нечітких множин.

4. Зміст розрахунково-пояснювальної записки:аналіз сучасних методів виявлення та запобігання взаємним блокуванням, розробка методу прогнозування взаємного блокування з урахування циклічного планування, розробка методу прогнозування взаємного блокування з урахування конкуренції процесів за використання ресурсів, розробка методу прогнозування взаємного блокування з урахування конкуренції між

процесами, розробка програмного забезпечення для методу прогнозування взаємних блокувань; економічна частина.

5. Перелік графічного матеріалу: організація циклічного планування; поетапне виконання системного виклику; обробка системного виклику; схема обробки системних викликів; композиція відносин між запитами від процесів та зверненнями до ресурсів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Хошаба О.М., к.т.н., доцент кафедр ПЗ		
4	Глущенко Л.Д., к.е.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
	Аналіз сучасних методів виявлення та запобігання взаємним блокуванням	07.09.2019 – 27.09-2019	Вик.
	Розробка методу прогнозування взаємного блокування	28.09.2019 – 28.10.2019	Вик.
	Розробка програмного забезпечення для методу прогнозування взаємних блокувань	29.10.2019 –15.11.2019	Вик.
	Економічна частина	16.11.2019 – 29.11.2019	Вик.

Студент _____ **Завертайло К.С.**

(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Хошаба О.М.**

(підпис) (прізвище та ініціали)

АННОТАЦІЯ

У магістерській кваліфікаційній роботі проведено ретельний аналіз методів і засобів запобігання ситуації взаємних блокувань і методи виявлення взаємних блокувань в операційній системі. Сформульовано мету досліджень – зменшення ризику виникнення ситуації взаємних блокувань в, а також зменшення навантаження на роботу планувальника в операційних системах.

В роботі пропонується метод прогнозування взаємних блокувань в операційних системах. Розглянемо напрями, методи вирішення задачі взаємних блокувань. Запропонований метод, на відміну від аналогів, не знижує ефективність роботи процесів та не вимагає виконання додаткових інструкцій при системному виклику в операційній системі. При роботі даного методу не виявлено негативного впливу на засоби міжпроцесної взаємодії. Робота планувальника в ядрі операційної системи не підлягає додатковим навантаженням при плануванні роботи процесів в операційній системі. Програмні засоби, що реалізовані на базі запропонованого методу значно знижують ризик виникнення ситуації взаємного блокування при цьому, не знижуючи ефективність роботи операційної системи.

Розроблено метод та програмний засіб для прогнозування ситуацій взаємних блокувань в операційній системі. Отримані в магістерській кваліфікаційній роботі наукові та практичні результати можна використати при розробці ядра операційної системи та алгоритму планування роботи процесів.

ВСТУП

Актуальність теми дослідження. Задача виявлення та запобігання ситуації взаємних блокувань між процесами в операційних системах є актуальної від початку створення перших багатозадачних систем до сьогоднішнього дня. Було запропоновано та використовується на практиці багато методів уникнення та запобігання взаємним блокуванням, але кожен метод має ряд певних недоліків. Зазвичай, основними недоліками застосованих методів є ускладнення планування в операційних системах; значне збільшення обсягу обробляючої інформації для кожного з процесів в операційній системі, з метою контролю конкуренції процесів за використання ресурсів; та збільшення часу проведення в стані очікування для низькопріорітетних процесів в операційній системі.

При застосування методів уникнення взаємних блокувань в ядра операційної системи виникає додаткове завдання для обробки доступу процесів до певних ресурсів і зменшення швидкодії виконання процесами заданих інструкцій. Також, важливо наголосити, що жоден метод не приносить бажаного результату, тому актуальність розглянутої задачі актуальна і на сьогоднішній день. Немало розробників операційних систем відають перевагу методам усунення взаємних блокувань. Розроблені алгоритми таким типом методів, передбачають роботу вже з виникненою проблемою.

Також, задача вирішення взаємних блокувань зростає прямо пропорційно зростанню можливостей обробки інформації процесорами, оскільки зі збільшенням можливості обробки інформації центральним процесором збільшується кількість процесів, які система може обслуговувати в багатозадачному режимі. Чим більша кількість запущених в операційній системі процесів, тим більший ризик виникнення ситуації взаємного блокування. З цього можна зробити висновок, що методи усунення та

запобігання ситуації взаємних блокувань з часом також втрачають свою актуальність, що викликає потребу в розробці нових методів за для вирішення даної задачі.

Альтернативою вище вказаним методам може стати розробка методу прогнозування взаємних блокувань в операційній системі. Програмні засоби, що будуть розроблені за вказаним методом, не будуть викликати додаткового навантаження на ядро операційної системи, та планувальника операційної системи. Також процеси, згідно з запропонованим методом, не будуть витрачати свій час в стані очікування, заради доступу до певних ресурсів, оскільки прогнозування взаємних блокувань не вимагає додаткової перевірки пріоритетів процесу для доступу до запитуваних даних, як цього вимагають методи запобігання.

Розроблений метод буде значно впливати на роботу ядра операційної системи, конкретних процесів та роботу планувальника лише в тому та тільки в тому випадку, коли буде виявлена загроза виникнення ситуації взаємного блокування. Також даний метод, при оцінці алгоритму планування та типу операційної системи, можна буде застосовувати для практичної любої операційної системи, що вказує на незалежність від якогось типу операційних системи, чи якої конкретної операційної системи.

Мета та завдання дослідження. Метою є підвищення ефективності роботи планувальника операційної системи та роботи процесів в операційній системі. Вирішення задачі взаємних блокувань за рахунок розробки методів і програмних засобів усунення ситуації взаємних блокувань. Для того, щоб досягти поставленої мети потрібно вирішити такі поставлені завдання:

- Аналіз сучасних методів виявлення та запобігання взаємним блокуванням
- Розробка методу прогнозування взаємного блокування з урахування циклічного планування
- Розробка методу прогнозування взаємного блокування з урахування конкуренції процесів за використання ресурсів

- Розробка методу прогнозування взаємного блокування з урахування конкуренції між процесами
- Розробка програмних засобів для методу прогнозування взаємних блокувань

Об’єкт дослідження – процес організації роботи процесів в операційних системах

Предмет дослідження – методи та засоби для організації роботи процесів в операційній системі

Методи дослідження. У процесі дослідження використовувались такі математичні методи: теорія множин та теорія нечітких множин для збереження даних про роботу процесів та ресурсів в операційній системі, метод приведення до фазифікації нечітких множин та метод дефазифікації нечітких множин для обчислення прогнозування ситуації взаємних блокувань, методи розробки планування в операційних системах та методи розробки ядра операційної системи.

Наукова новизна одержаних результатів полягає в наступному:

- Вперше розроблено метод, особливість якого полягає в прогнозуванні ситуації взаємних блокувань, що підвищує ефективність роботи планувальника операційної системи
- Отримано аналітичні залежності роботи планувальника від організації процесів в операційній системі, що забезпечило підвищення швидкодії роботи процесів в операційній системі

Практичне значення одержаних результатів полягає в тому, що на основі проведених теоретичних досліджень і отриманих наукових результатів розроблено алгоритми та програмний засіб для прогнозування ситуації взаємних блокувань в операційній системі, зокрема, внесено зміни в програмні засоби, що відповідають за системні виклики, внесено зміни в програмні засоби, що відповідають за роботу планувальника в операційній системі.

Достовірність теоретичних положень підтверджується чіткістю та строгістю постановки задач для магістерської кваліфікаційної роботи. Під час

доведення наукових положень були використані коректні математичні методи, строго виведено аналітичні співвідношення, виконано порівняння результатів з відомими, та перевірена збіжність результатів математичного моделювання з результатами, що отримані під час впровадження розроблених програмних засобів.

Особистий внесок магістранта. Результати, що наведені у магістерській кваліфікаційній роботі, були отримані магістрантом самостійно.

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ВИЯВЛЕННЯ ТА ЗАПОБІГАННЯ ВЗАЄМНИМ БЛОКУВАННЯМ

1.1. Умови виникнення взаємних блокувань

В комп'ютерних системах налічується багато ресурсів і тільки один процес може використовувати конкретний ресурс у конкретний час. Дуже часто виникає ситуація, коли два і більше процеса подають запит для того, щоб використовувати один і той самий ресурс. Наприклад, з даним ресурсом працює процес під номером 1, опрацювавши його, він подає запит на використання пам'яті системи і при цьому залишає ресурс у своєму розпорядженні, тому що він планує після роботи з пам'яттю, продовжити роботу з даним ресурсом. В цей час планувальник вирішує, що процес під номером один достатньо використав процесорний час, та забирає в нього можливість роботи з процесором за допомогою переривання, і після того як перший процес виконав операції з пам'яттю, переходить у режим готовності, очікуючи коли йому буде надано можливість працювати. Далі час роботи надається процесу під номером два і для роботи з процесором, другому процесу потрібно використовувати той самий ресурс, яким щойно користувався процес під номером один, але даний ресурс знаходиться у розпорядженні першого процесу, який, в свою чергу, не збирається віддавати можливість роботи з цим ресурсом іншим процесам, адже він планує його використовувати далі.

Таким чином другий процес використає весь виділений планувальником йому час так і простоїть в очікуванні на використовування даного ресурсу. І сам процес простоїть «в пусту», тобто знаходячись у стані очікування. Так, через певний проміжок часу, планувальник вирішить виділити час першому процесу у даній ситуації, але не факт, що буде вибрано процес під номером один, адже в операційних системах є багато процесів, які конкурують за процесорний час і вони також можуть подати запит на використання даного

ресурсу і потрапити у таку ж саму ситуацію, які і другий процес.

Приведений вище приклад є одним з різновидів взаємних блокувань. Ця проблема постає дуже гостро в розробках операційних системах, оскільки в таких ситуаціях процеси знаходяться в очікуваннях, а не працюють, а процесор простоюється і не використовується. І якщо, ситуація критична, якщо багато процесів потрапили у ситуацію взаємних блокувань, то значно зменшиться продуктивність роботи самої системи.

Для вирішення даної проблеми було запропоновано немало методів вирішення, але всі вони мають свої недоліки. Розглянемо методи виявлення взаємних блокувань.

Найчастіше для боротьби з взаємними блокуваннями використовуються чотири стратегії:

- Ігнорування проблеми
- Виявлення та відновлення. Дати взаємним блокуваннями проявити себе задля їх подальшого усунення
- Динамічне ухилення від них за рахунок ретельного розподілу ресурсів
- Запобігання за рахунок структурного усунення одного з чотирьох умов, необхідних для їх виникнення.
-

1.2. Розгляд ресурсного взаємного блокування

Взаємне блокування в групі процесів виникає в тому випадку, якщо кожен процес з цієї групи очікує події, настання якого залежить виключно від іншого процесу з цієї ж групи.

Оскільки всі процеси перебувають у стані очікування, жоден з них не стане причиною якої-небудь події, яке могло б відновити роботу іншого процесу, що належить до цієї групи, і очікування всіх процесів стає нескінченним. У цій моделі передбачається, що у процесів є тільки один потік, а переривання, здатні відновити роботу заблокованого процесу, відсутні. Умова відсутності переривань необхідно, щоб не дозволити заблокованого з інших

причин процесу відновити свою роботу з аварійного сигналу, після чого викликати подія, що звільняє інші наявні в групі процеси. У більшості випадків подією, настання якого очікує кожен процес, є вивільнення будь-якого ресурсу, яким на даний момент володіє інший учасник групи. Іншими словами, кожен процес з групи, що потрапила в ситуацію взаємне блокування, очікує ресурсу, яким володіє інший процес з цієї ж групи. Жоден з процесів не може працювати, жоден з них не може вивільнити будь-якої ресурс, і жоден з них не може відновити свою роботу. Кількість процесів і кількість і вид утримуваних і запитуваних ресурсів не має значення. Цей результат зберігається для будь-якого типу ресурсів, включаючи апаратні і програмні ресурси. Цей вид взаємного блокування називається ресурсним взаємним блокуванням.

Взаємні блокування при обміні даними є ще одним різновидом взаємних блокувань, яке може проявитися в системах обміну даними, наприклад, мережах, в яких один і більше процесів зв'язуються шляхом обміну повідомленнями. Загальна домовленість передбачає, що процес А відправляє повідомлення-запит процесу В, а потім блокується до тих пір, поки В не надішле назад у відповідь. Припустимо, що повідомлення-запит десь не дійшло до призначеного процесу. Процес А заблокований в очікуванні відповіді. Процес В заблокований в очікуванні запиту на будь-які його дії. В результаті виникає взаємне блокування.

Взаємне блокування, що пов’язане з обміном даних не є класичним як те, що пов'язане з ресурсами. Процес А не володіє якимись ресурсами, які потрібні процесу В, і навпаки. Фактично ніяких ресурсів тут немає. Але тим не менше це взаємне блокування, відповідна формальному визначенню, оскільки є група з двох процесів, кожен з яких заблокований, чекаючи події, причиною якого може бути тільки інший процес. Описану ситуацію назвали комунікаційним взаємним блокуванням, щоб відрізнити її від більш поширеним на практиці ресурсним взаємним блокуванням. Комунікаційне взаємне блокування є аномальним варіантом синхронізації спільних дій. Процес в цьому типі

взаємного блокування не може завершити обслуговування, якщо виконується незалежно від інших процесів.

Комунікаційні взаємні блокування не можуть бути запобігти за рахунок упорядкування ресурсів (за відсутністю таких) або обійдені за рахунок ретельного планування (оскільки тут немає моментів, коли запит може бути відкладений). Проте, існує інша технологія, яка зазвичай може бути застосована для припинення комунікаційного взаємного блокування, - витікання часу очікування. У більшості мережевих систем обміну даними, як тільки послано повідомлення, на яке очікується відповідь, відразу запускається таймер. Якщо виставлене в таймері час закінчиться до надходження відповіді, відправник повідомлення припустить, що повідомлення не надійшло, і відішле його знову (і знову, і знову, якщо потрібно). Таким чином взаємне блокування буде попереджено.

Рисунок 1.1 Ресурсне взаємне блокування

Коли пакет надходить в маршрутизатор від одного з вузлів, він поміщається в буфер для подальшої передачі іншому маршрутизатора, потім

наступного, до тих пір поки не буде переданий за призначенням. Ці буфери є ресурсами, і їхня кількість кінцевим числом. На рис. 1.1 у кожного маршрутизатора є тільки вісім буферів (на практиці у них є мільйони буферів, але це не змінює суті потенційного взаємного блокування, а тільки впливає на частоту її виникнення). Припустимо, що всі пакети з маршрутизатора A потрібно передати маршрутизатора B, всі пакети з маршрутизатора B повинні бути відправлені на маршрутизатор C, всі пакети з C повинні піти до D, а все пакети з D повинні піти до A. Але жоден пакет не може бути переміщений, оскільки на іншому кінці немає вільного буфера і ми маємо справу з класичною ресурсним взаємним блокуванням, хоча і в центрі комунікаційної системи.

1.3. Моделювання взаємних блокувань

Застосуємо в моделюванні направлені графи. У графів є два види вузлів: процеси, показані колами, і ресурси, показані квадратами. Направлене ребро, яке слід від вузла ресурсу (квадрата) до вузла процесу (кола), означає, що цей ресурс був раніше запитано, отриманий і на даний момент утримується цим процесом. На рис. 1.2, а ресурс R в даний час виділено процесу A.

Направлене ребро, що йде від процесу до ресурсу, означає, що процес в даний час заблокований в очікуванні вивільнення цього ресурсу. На рис. 1.2, процес B очікує вивільнення ресурсу S. На рис. 1.2, в ми спостерігаємо взаємне блокування: процес C очікує вивільнення ресурсу T, який в даний момент утримується процесом D. Процес D не збирається вивільняти ресурс T, оскільки він очікує вивільнення ресурсу U, утримуваного процесом C. Обидва процеси знаходяться в стані вічного очікування. Циклічна структура графа означає, що ми маємо справу з взаємним блокуванням, що включила процеси і ресурси в цикл (передбачається, що в системі є тільки один ресурс кожного типу). В даному прикладі вийшов наступний цикл: C-T-D-U-C.

Такий порядок не призводить до взаємне блокування (оскільки відсутня боротьба за оволодіння ресурсами), але в ньому немає і ніякої паралельної

роботи. Крім дій за запитом і вивільненню ресурсів процеси займаються ще й обчисленнями, і введенням-виведенням даних. Коли процеси запускаються послідовно, відсутня можливість використання центрального процесора одним процесом, в той час коли інший процес чекає завершення операції введення-виведення. Тому строго послідовне виконання процесів може бути не оптимальним рішенням. У той час, якщо жоден з процесів не виконує ніяких операцій введення-виведення, алгоритм, при якому найкоротший завдання виконується першим, більш привабливий, ніж циклічний алгоритм, тому за певних умов послідовний запуск всіх процесів може бути найкращим рішенням.

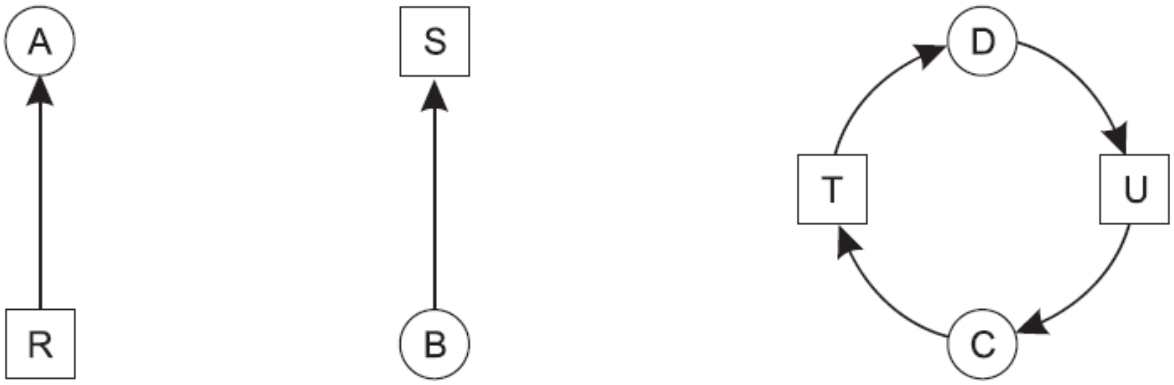

Рисунок 1.2 Граф розподілу ресурсів.

Припустимо, що процеси займаються як введенням-висновком, так і обчисленнями, тому найбільш розумним алгоритмом планування їх роботи є циклічний алгоритм. Запити ресурсів можуть відбуватися в тому порядку, який показаний на рис. 1.3. Якщо ці шість запитів виконуються саме в цьому порядку, їм відповідаю шість результуючих ресурсних графів, показаних на рис. 2.3. Після видачі запиту процес А, як показано на рис. 1.3, блокується в очікуванні ресурсу S. На наступних двох етапах процеси В і С також блокуються, що в результаті призводить до зациклення і виникненню взаємного блокування, показаної на рис.1.3. Якщо операційна система знає про прийдешню взаємне блокування, вона може призупинити процес В, замість

того щоб виділити йому ресурс S. Запускаючи тільки процеси A і C, ми отримаємо таку послідовність дій за запитом і вивільненню ресурсів. Після закінчення етапу, показаного на рис. 1.3, процесу B може бути виділений ресурс S, оскільки процес A завершив свою роботу, а процес C має все необхідне. Навіть якщо B заблокується при запиті ресурсу T, взаємне блокування не виникне. Процес B просто чекатиме, поки процес C не завершить свою роботу.

У інформатиці взаємного блокування посилається на конкретну умову, коли два або більше процесів чекають, коли інший випустить ресурс, або більше двох процесів очікують на ресурси в круговому ланцюжку. Взаємне

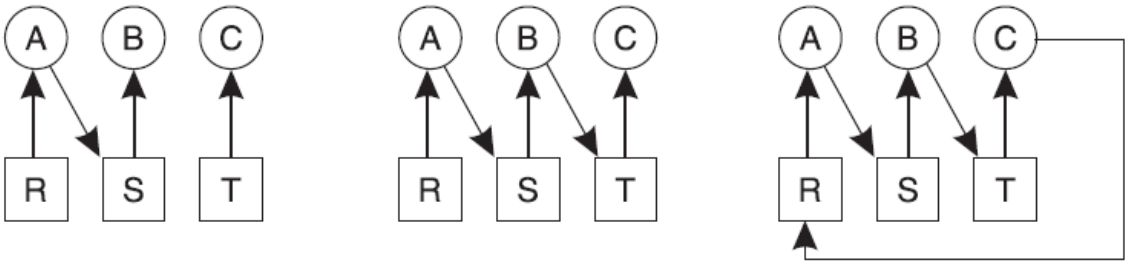

Рисунок 1.3 Приклад виникнення взаємного блокування

блокування - це поширена проблема багатопроцесорної обробки, коли багато процесів поділяють певний тип взаємовиключного ресурсу, відомого як програмне забезпечення, або м'якого блокування. Комп'ютери, призначені для ринків обміну часу в режимі реального часу, часто оснащені апаратним блокуванням або жорстким блокуванням, що гарантує ексклюзивний доступ до процесів, що примушує серіалізацію. Взаємні блокування особливо перешкоджають нормальній роботі, оскільки не існує загального рішення, щоб уникнути м'яких взаємних блокувань.

Цю ситуацію можна зрозуміти за аналогією з класичною в інформатиці задачею про обідаючих філософів, які одночасно беруть вилки з однієї сторони і не можуть взяти вилку з іншої сторони, оскільки вона зайнята сусідом. Тому

кожен з філософів знаходиться у стані очікування, допоки сусід не звільнить вилку. Всі запити не можуть бути задоволені, тому виникає тупикова ситуація.

Опис телекомунікаційного зв'язку з тупиком трохи сильніше: тупикова ситуація виникає, коли жоден з процесів не відповідає умові переходу в інший стан (як описано в машині кінцевого стану процесу) і всі канали зв'язку порожні. Друга умова часто залишається поза іншими системами, але є важливою в телекомунікаційному контексті та його системах.

Існують чотири необхідні умови для виникнення тупикової ситуації:

- умова взаємного виключення: ресурс не може бути використаний більш ніж одним процесом одночасно
- умова затримки і очікування: процеси, в яких вже зберігаються ресурси, можуть вимагати нових ресурсів
- немає умови попередження: лише процес, що містить ресурс, може випустити його
- умова кругового очікування: два або більше процесів утворюють круговий ланцюг, де кожен процес чекає ресурсу, який має наступний процес у ланцюзі

Взаємне блокування може виникнути лише в системах, де всі 4 умови справджуються. Профілактика кругового очікування полягає у тому, щоб дозволити процесам чекати ресурси, але гарантувати, що очікування не може бути круговим. Одним із підходів може бути присвоєння пріоритету кожному ресурсу та змушення процесів запитувати ресурси в порядку збільшення пріоритетності. Тобто, якщо процес містить деякі ресурси, і найвищий пріоритет цих ресурсів становить m, то цей процес не може вимагати жодного ресурсу з перевагою менше m. Це змушує розподіл ресурсів виконувати певний та не круговий упорядкування, тому кругового очікування не може відбутися. Інший підхід - дозволити мати лише один ресурс за процес; якщо процес вимагає іншого ресурсу, він повинен спочатку звільнити той, який він зараз тримає або перейти у стан готовності.

Приклад взаємного блокування, який може статися в продуктах бази даних, наступний. Клієнтські програми, що використовують базу даних, можуть вимагати ексклюзивного доступу до таблиці, а для отримання ексклюзивного доступу вони вимагають блокування. Якщо одна клієнтська програма тримає замок на столі і намагається отримати замок на другій таблиці, яка вже утримується другою клієнтською програмою, це може призвести до тупикової ситуації, якщо друга програма намагається отримати замок, який утримує перше застосування, але цей конкретний тип взаємного блокування легко запобігається, наприклад, за допомогою алгоритму розподілу ресурсів, що повністю або немає.

Іншим прикладом може бути програма для форматування тексту, яка приймає відправлений до нього текст на обробку і потім повертає результати, але робить це лише після отримання «достатнього» тексту для роботи. Написана програма редактора тексту, яка надсилає форматору з деяким текстом, а потім чекає результатів. У цьому випадку на останньому блоці тексту може статися тупик. Оскільки у форматора може бути недостатньо тексту для обробки, він призупинить себе під час очікування додаткового тексту, який ніколи не надійде, оскільки текстовий редактор надіслав його всім текстом, який він має. Тим часом сам текстовий редактор призупиняється в очікуванні останнього виводу з форматора. Цей тип взаємного блокування іноді називають смертельним обіймами (належним чином використовується лише тоді, коли задіяні лише два додатки) або голодуванням. Однак цю ситуацію також легко запобігти, якщо текстовий редактор надішле повідомлення про форсування (наприклад, EOF) з його останнім (частковим) блоком тексту, що змусить форматник повернути останній (частковий) блок після форматування, і не чекати додаткового тексту.

Тим не менше, оскільки не існує загального рішення щодо запобігання тупиковій ситуації, кожен тип взаємного блокування ситуації повинен бути передбачений і спеціально запобігати. Але загальні алгоритми можуть бути

реалізовані в операційній системі, так що якщо одне або більше додатків заблокується, воно, як правило, після закінчення припиняється (а тим часом не

Рисунок 1.4 Схема роботи операційної системи

допускається жодних інших ресурсів і може знадобитися здати ті, які вже є, відкотившись назад до стану до отримання заявою).

Взаємне блокування можна уникнути, якщо певна інформація про процеси буде доступна заздалегідь щодо розподілу ресурсів. Для кожного запиту на ресурс система бачить, якщо надання запиту означатиме, що система

перейде в небезпечний стан, тобто стан, який може призвести до тупикової ситуації. Тоді система лише надає запити, що призведуть до безпечних станів. Для того щоб система змогла розібратися, чи буде наступний стан безпечним чи небезпечним, вона повинна заздалегідь знати в будь-який час кількість та тип усіх наявних ресурсів, наявних та запитуваних. Одним з відомих алгоритмів, який використовується для запобігання взаємного блокування, є алгоритм Банкіра, який вимагає заздалегідь знати ліміт використання ресурсів. Однак для багатьох систем неможливо заздалегідь знати, що вимагатиме кожен процес. Це означає, що уникнення тупикових ситуацій часто неможливо.

Ще два алгоритми – очікування/завершення та пошкодження/очікування, кожен з яких використовує техніку розриву симетрії. В обох цих алгоритмах існує старший процес (O) і молодший процес (Y). Вік процесу можна визначити за допомогою позначки часу на час створення процесу. Менші часові позначки - це старіші процеси, тоді як більші часові позначки - молодші.

Важливо зауважити, що процес може перебувати в небезпечному стані, але не призведе до тупикової ситуації. Поняття безпечного / небезпечного стану стосується лише здатності системи переходити у стан взаємного блокування чи ні. Наприклад, якщо процес запитує A, що призведе до небезпечного стану, але випускає B, що перешкоджатиме круговому очікуванню, стан не є безпечним, але система не знаходиться у взаємному блокуванні.

Взаємне блокування можна запобігти, забезпечивши, щонайменше, одну з наступних чотирьох умов:

Видалення умови взаємного виключення означає, що жоден процес не може мати виключний доступ до ресурсу. Це виявляється неможливим для ресурсів, які неможливо зібрати, і навіть із запущеними ресурсами все-таки може виникнути взаємне блокування. Алгоритми, що уникають взаємного виключення, називаються алгоритмами, що не блокують синхронізацію.

Умови «затримати і чекати» можуть бути усунені, вимагаючи від процесів вимагати всі необхідні ресурси перед запуском (або перед початком певного набору операцій); цим заздалегідь знанням часто важко задовольнятись

і, у будь-якому випадку, це неефективне використання ресурсів. Інший спосіб - вимагати від процесів звільнення всіх своїх ресурсів, перш ніж вимагати всі необхідні їм ресурси. Це теж часто недоцільно. Такі алгоритми, такі як маркери серіалізації, відомі як алгоритми «всі або ні».

Умову «без попередження» (блокування) також може бути важко або неможливо уникнути, оскільки процес повинен мати можливість мати ресурс протягом певного часу, або результат обробки може бути невідповідним або може трапитися подрібнення. Однак неможливість застосування превенції може заважати алгоритму пріоритетності. Прийняття «заблокованого» ресурсу, як правило, передбачає відкат, і його слід уникати, оскільки це дуже дорого обходиться. Алгоритми, що дозволяють дозволити, включають алгоритми без блокування та без очікування та оптимістичний контроль одночасності.

Умова кругового очікування: Алгоритми, які уникають кругового очікування, включають «відключення переривань під час критичних розділів» та «використання ієрархії для визначення часткового впорядкування ресурсів» де немає явної ієрархії, для визначення навіть використовували адресу пам'яті ресурсів і рішення Дійкстри.

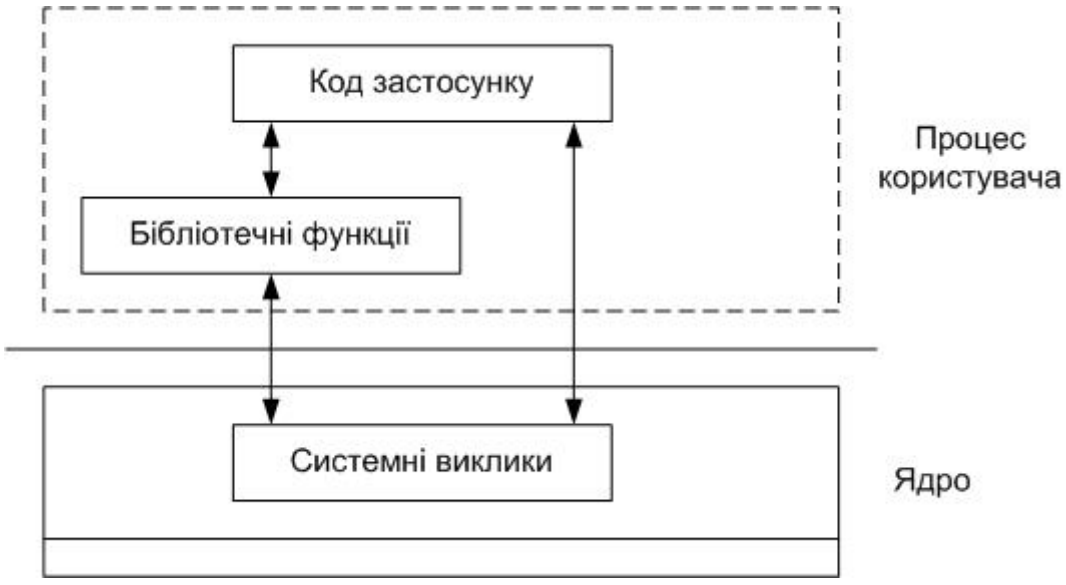

Рисунок 1.5 Схема системного виклику

Часто не можна застосовувати ні запобігання взаємного блокування, ні запобігання блокування. Натомість виявлення взаємного блокування та перезапуск процесу використовуються за допомогою алгоритму, який відстежує розподіл ресурсів та стани процесу, а також відкочує та перезапускає один або декілька процесів, щоб усунути взаємного блокування. Виявити вже зафіксований тупик легко, оскільки ресурси, які кожен процес заблокував та / або наразі запитуються, відомі планувальнику ресурсів або операційної системи.

Виявити можливість взаємного блокування до його виникнення набагато складніше і насправді, як правило, не можна визначити, оскільки проблема зупинки може бути переосмислена як сценарій взаємного блокування. Однак у конкретних середовищах, використовуючи певні засоби блокування ресурсів, виявлення зв'язку взаємного блокування може бути вирішеним. У загальному випадку не можна відрізнити алгоритми, які просто чекають настання дуже малоймовірної сукупності обставин, та алгоритми, які ніколи не закінчаться через взаємного блокування.

1.4. Методи запобігання та усунення взаємних блокувань

Технологія виявлення і відновлення. При її використанні система не намагається запобігати взаємні блокування. Вона дозволяє їм відбутися, намагається виявити момент їх виникнення, а потім робить деякі дії по відновленню працездатності. У цьому розділі будуть розглянуті деякі способи виявлення взаємних блокувань і деякі методи відновлення працездатності, які можна реалізувати.

Виявлення взаємного блокування при використанні одного ресурсу кожного типу. Для такої системи можна побудувати ресурсний граф Якщо цей граф містить один і більше циклів, значить, ми маємо справу з взаємним блокуванням. Будь-який процес, є частиною циклу,який заблокований намертво. Якщо циклів немає, значить, система не перебуває в стані взаємного

блокування.

Цей алгоритм бере по черзі кожен вузол в якості кореневого в надії, що з цього вийде дерево, і виконує в дереві пошук в глибину. Якщо в процесі обходу алгоритм повертається до вже зустрічалися вузлу, значить, він знайшов цикл. Якщо алгоритм обходить всі ребра з якого-небудь заданого вузла, то він повертається до попереднього вузла. Якщо він повертається до кореневого вузла і не може йти далі, то підграф, доступний з поточного вузла, не містить циклів. Якщо ця властивість зберігається для всіх вузлів, значить, повний граф не містить циклів, а система не знаходиться в стані взаємного блокування. Виявлення взаємних блокувань при використанні декількох ресурсів кожного типу. Кожен процес спочатку оголошується немаркованих. У міру роботи процеси позначатимуться, показуючи, що вони здатні завершити свою роботу і не беруть участі у взаємному блокуванні. Коли алгоритм завершує свою роботу, будь непомічений процес вважається які беруть участь у взаємному блокуванні. При роботі цього алгоритму передбачається найгірший з можливих сценаріїв розвитку подій: всі процеси утримують всі отримані ресурси до тих пір, поки не закінчать свою роботу. На першому кроці алгоритм шукає процес, який може допрацювати до кінця. Такий процес характеризується тим, що всі його запити на ресурси можуть бути задоволені за рахунок поточних доступних ресурсів. Тоді обраний процес допрацює до кінця, після чого поверне все утримувані їм ресурси в фонд доступних ресурсів. Потім цей процес позначається завершеним. Якщо в результаті виявиться, що все процеси можуть допрацювати до кінця, значить, жоден з них не бере участь у взаємному блокуванні. Якщо частина процесів ніколи не зможе допрацювати до кінця, значить, вони знаходяться в стані взаємному блокуванні. Хоча алгоритм не є детермінованим (оскільки він може запускати процеси в будь-якому можливому порядку), результат завжди однаковий.

Механізм продуктів Business Process Engine Choreographer бізнес-процесів (BPC) зберігається в базі даних BPC, інформація про всі процеси та обох екземплярах задає довгих виконуючих процесів. Це можливо для

користувальницьких додатків, виконуючи механізми Business Process Engine, визначити трансакції, одночасне виконання. Це може бути представлено у тому випадку, якщо додаток користувача задає дуже тривалі транзакції, які оновляють багато ресурсів.

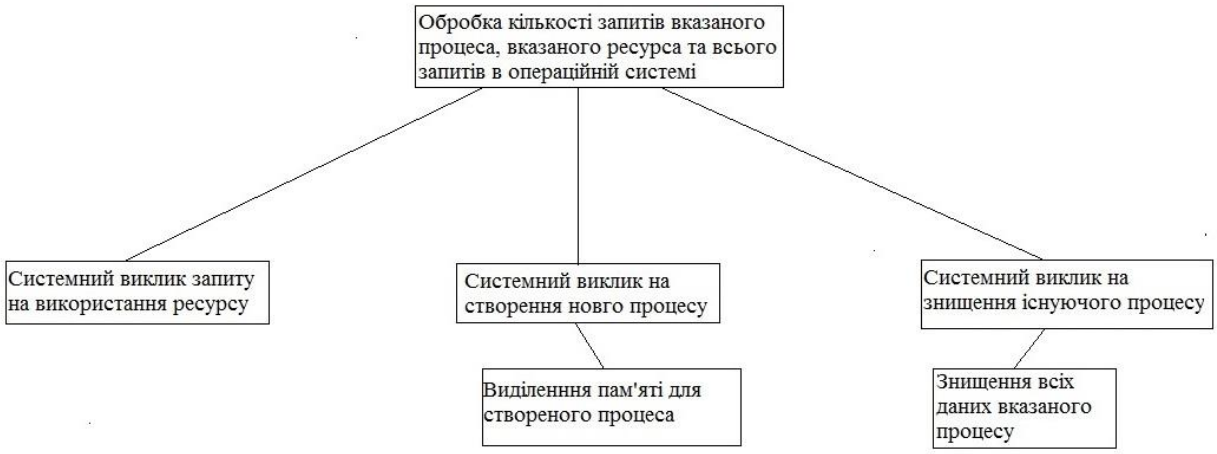

Рисунок 1.6 Схема обробки кількості звернень в системі

Завдяки механізму виявлення та усунення взаємних блокувань кожна взаємне блокування в кінцевому підсумку усувається без негативного впливу на функціональні можливості програми. Постраждати може лише продуктивність - якщо взаємні блокування відбуваються занадто часто. Якщо справа йде так само, може допомогти настройка бази даних і системи, хоча в деяких випадках потрібно оптимізація програми. Зазвичай взаємні блокування виявляються на етапі тестування програми - при виконанні паралельних тестів і при збільшенні тестового навантаження. Чим вище рівень паралелізму і чим більше тривалість транзакції, тим вище ймовірність виникнення взаємних блокувань і тайм-аутів блокування.

У досить складній системі, що налічує десятки різноманітних типів бізнес транзакцій, навряд чи вийде спроектувати все транзакції таким чином, щоб взаємне блокування не може виникнути ні за яких умов. Не варто витрачати час на запобігання взаємних блокувань, ймовірність виникнення яких украй мала. Але, щоб не псувати user experience, в разі, коли операція переривається через

взаємного блокування, її потрібно повторити. Для того, щоб операцію можна було безпечно повторити, вона не повинна змінювати вхідні дані і має бути загорнута в одну транзакцію (або замість всієї операції, треба обертати на свою RetryOnDeadlock кожну SQL транзакцію в операції). Важливо розуміти, що функція RetryOnDeadlock всього лише покращує user experience при зрідка виникають взаємних блокувань. Якщо вони виникають дуже часто, вона лише погіршить ситуацію, в рази збільшивши навантаження на систему.

Критичні секції. Важливим поняттям синхронізації потоків для вирішення проблеми змагань є поняття критичної секції програми. Критична секція - це частина програми, результат виконання якої може непередбачуваної змінюватися, якщо змінні, що відносяться до цієї частини програми, змінюються іншими потоками в той час, коли виконання цієї частини ще не завершено. У всіх потоках, які працюють з критичними даними, повинна бути визначена критична секція. У різних потоках критична секція складається в загальному випадку з різних послідовностей команд. Найпростіший і в той же час найменш ефективний спосіб забезпечення взаємного виключення полягає в тому, що ОС дозволяє потоку забороняти будь-які переривання на час його перебування в критичній секції. Однак цей спосіб практично не застосовується, так як небезпечно довіряти управління системою призначеному для користувача потоку - він може надовго зайняти процесор, а у випадку краху потоку в критичній секції крах зазнає вся система, тому що переривання ніколи не будуть вирішені. Умови виключення гонок Два процеса не повинні одночасно перебувати в критичній секції У програмі не повинно бути припущень про швидкість або кількості процесорів. Процес поза критичної секції не може блокувати інші процеси. Слід унеможливити ситуація, коли процес вічно чекає попадання в критичну секцію. Взаємне виключення з використанням критичних секцій

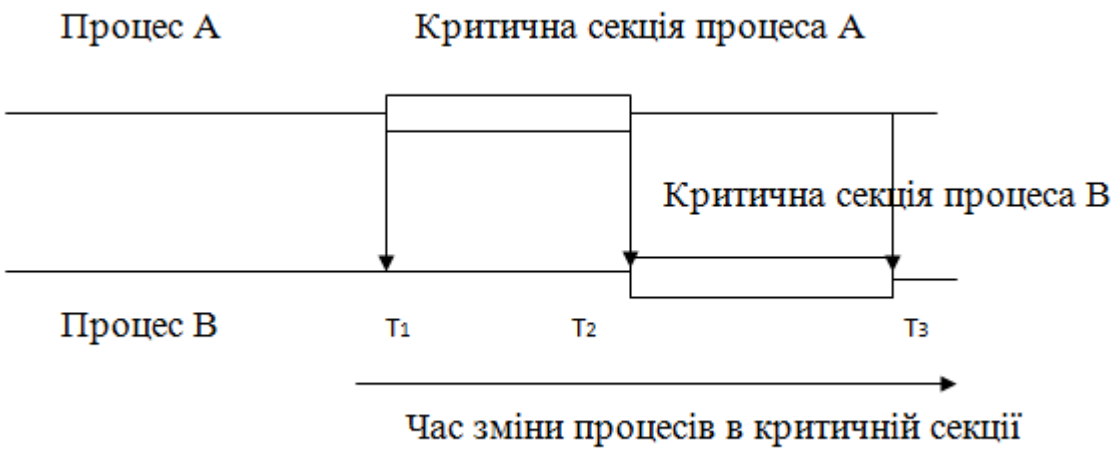

Рисунок 1.7 Критичні секції

Семафор - невід'ємна ціла змінна S> = 0, яка може змінюватися і перевірятися тільки за допомогою двох примітивів: V (S): змінна S збільшується на 1 єдиним неподільним дією. До змінної S немає доступу іншим потокам під час виконання цієї операції. P (S): зменшення S на 1, якщо це можливо. Якщо S = 0 і неможливо зменшити S, залишаючись в області цілих невід'ємних значень, то в цьому випадку потік, що викликає операцію P, чекає, поки це зменшення стане можливим. Успішна перевірка і зменшення також є неподільною операцією.

Потік отримує доступ до ресурсу, викликаючи одну з Wait-функцій і передаючи їй опис семафора, який охороняє цей ресурс Wait-функція перевіряє у семафора лічильник поточного числа ресурсів якщо його значення більше 0 (семафор вільний), зменшує значення цього лічильника на 1, і викликає потік залишається планованим.

Якщо Wait-функція визначає, що лічильник поточного числа ресурсів дорівнює 0 (семафор зайнятий), система переводить викликає потік в стан очікування Коли інший потік збільшить значення цього лічильника, система згадає про режимі потоці і знову почне виділяти йому процесорний час (а він, захопивши ресурс, зменшить значення лічильника на 1).

Іноді використовується спрощена версія семафора - м'ютекс. Іноді називають ще двійковим семафором. М'ютекс - змінна, яка може перебувати в одному з двох станів: блокованому або неблокованому. Якщо процес хоче увійти в критичну секцію - він викликає примітив блокування мьютекса. Якщо м'ютекс не заблокований, то запит виконується і процес потрапляє в критичну секцію. У розглянутому прикладі, для того щоб виключити колізії при роботі з розділяється областю пам'яті, будемо вважати, що запис в буфер і зчитування з буфера є критичними секціями. Взаємне виключення будемо забезпечувати за допомогою двійкового семафора (мьютекса). Обидва потоки після перевірки доступності буферів повинні виконати перевірку доступності критичної секції.

Якщо ж в ньому передається TRUE, ідентифікатор потоку, що належить м'ютексів, прирівнюється ідентифікатором викликає потоку, а лічильник рекурсії отримує значення 1. Оскільки тепер ідентифікатор потоку відмінний від 0, м'ютекс спочатку знаходиться в зайнятому стані.

Потік отримує доступ до ресурсу, викликаючи одну з Wait-функцій і передаючи їй описатель мьютекса, який охороняє цей ресурс. Wait-функція перевіряє у мьютекса ідентифікатор потоку, якщо згор значення не дорівнює 0, м'ютекс вільний, в іншому випадку воно приймає значення ідентифікації потоку, і цей потік залишається планованим.

Рисунок 1.8 Зміна станів процеса

Всі розглянуті методи щодо запобігання взаємних блокувань ніяк не пов’язанні з їх прогнозуванням. Основна ідея даних методів є або введення додаткових змінних або функцій заради перевірки та перестрахування, що в цілому не застраховує процесів від взаємних блокувань. Оскільки розробка методів уникнення проблеми взаємних блокувань є дуже важкою, тому що складно урахувати та розподілити час роботи процесу, тим більш дані методи не враховують час стану в якому провів той чи інший процес, що перед цим методом робить рівними абсолютно всі процеси, навіть фонові.

Методи, який пропоную уникнути взаємні блокування заборонами входити в часте блокування про конкретного процеса та введення критичних секцій, значно обмежують роботу процесів. В таких ситуаціях процеси опиняються в додатковій стадії очікування, що спричиняє значні витрати та іноді лишнє простоювання процесора і в цілому, значно гальмує час виконання заданих функцій перед кожним процесом.

Встановлення шляхом припинення роботи процесів. Найпростішим способом перервати взаємні блокування є усунення одного або декількох процесів. Можна знищити процес,що знаходиться в циклі взаємного блокування. Якщо пощастить, то інші процеси зможуть продовжити свою роботу. Якщо це не допоможе, то все можна повторити, поки цикл не буде розірваний.

В якості альтернативи можна вибрати жертвою процес, що не знаходиться в циклі, щоб він вивільнив утримувані їм ресурси. При цьому підході знищуваний процес вибирається з особливою ретельністю, тому що він повинен утримувати ресурси, необхідні деяким процесам в циклі. Наприклад, один процес може утримувати принтер і вимагати плоттер, а інший, навпаки, утримувати плоттер і запитувати принтер. Обидва вони знаходяться в стані взаємного блокування. Третій процес, в свою чергу може утримувати такий самий плоттер і також такий самий принтер і, при цьому, цілком нормально виконувати свою роботу. І саме негайне завершення роботи третього процеса приведе до того, що всі ресурси, які він займає, стануть вільними для

використання іншими процесами і, саме головне, розірве ланцюг взаємного блокування між попередніми двома процесами.

По можливості краще вбити процес, який може бути безболісно перезавантажений з самого початку. Наприклад, компіляція завжди може бути перезавантажено, оскільки все, що вона робить, - це читає вхідний файл і створює об'єктний файл. Даний метод далеко не є ідеальним, оскільки в ньому не оцінюється активність процесів, дуже важливо відрізняти процеси, що залежать від ресурсів, від тих, які залежать від використання процесорного часу. В ньому немає оцінки кількості часу проведення процесів у тому чи іншому стані, також коли оцінюється вже займаний ресурс, а не ресурс до якого щойно було звернення від певного процеса, то шанс уникнути або попередити ситуацію взаємного блокування значно знижений, адже, як було розглянуто, є невигружаємі ресурси, і припинити роботу процеса, який зайняв саме такі

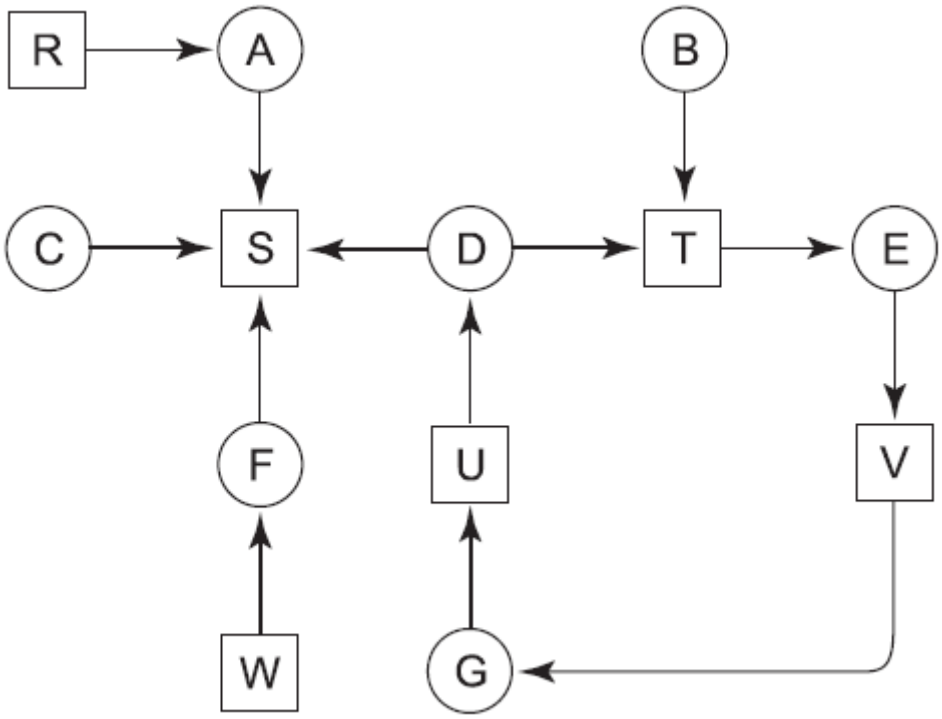

Рисунок 1.9 Виявлення взаємного блокування

ресурси є значною проблемою, яка обов’язково скажеться на працездатності операційної системи в даний час.

1.5. Прогнозування взаємних блокувань

Всі, розглянуті методи, мають один значний недолік – вони застосовуються для виходу з вже виникненої ситуації взаємних блокувань, або це методи запобігання взаємним блокуванням. Методи усунення взаємних блокувань працюють вже тоді, коли саме проблема, безпосередньо, виникла та перед ними постає питання найбільш доцільного завершення роботи певних процесів, найбільш коректного завершення ресурсів. Найпростіший алгоритм, при якому просто знищується певна кількість процесів, оскільки дуже рідко буває таке, що при значній кількості процесів, що потрапили у ланцюг взаємного блокування, потрібно негайно завершити роботу лише одного процеса, є дуже грубим по відношенню до працюючої системи, при чому це є певне навантаження на планувальника, особливо якщо метод розстановки пріоритетів, яким користувався планувальник, взаємопов’язує, наприклад, оцінку виділеного часу для кожного в системі процеса.

Методи запобігання взаємним блокуванням також мають суттєві недоліки, які заключаються в тому, що при розробці методів такого типу, завжди жертвують працездатністю системи. Наприклад, введення критичних секцій, значно ускладнює обробку даних введення-виводу оскільки потрібно розраховувати скільки процесів подають запит на той чи інший ресурс. Також основним недоліком є те, що процеси, які подають запит на використання одного і того ж ресурсу вимушені очікувати в черзі поки лише один процес завершить свою роботу та перестане утримувати ресурс, тим самим звільнить критичну секцію. Це також вимагає введення додаткового алгоритму перевірки черги процесів перед тою чи іншою критичною секцією.

Введення семафорів та мютексів в повній мірі не вирішило проблему. Оскільки для цього знову ж таки, потрібно слідкувати за коректною роботу

алгоритмів, що відслідковують роботу процесів, відносно цих змінних, та призначення біта доступу цим змінним. Також це породило певну рекурсію щодо взаємних блокувань, тобто взаємні блокування між процесами можуть відбутися безпосередньо при роботі як і з семафорами та і з мютексами.

Альтернативою для методів усунення та запобігання взаємним блокуванням є прогнозування взаємних блокувань в операційній системі. Розробка методу прогнозу такої ситуації значно зменшить ризик виникнення взаємного блокування, та не буде потребувати додаткових звернень до специфічних змінних та введення критичних секцій, а це в в сою чергу, значно збільшить працездатність всіх процесів, адже їм не потрібно витрачати час на певні перевірки та занесення і зчитування даних.

Також використання методу прогнозування ситуації взаємних блокувань не буде безпосередньо впливати на роботу самої системи, окрім випадків, коли програмний засіб прогнозування ситуації взаємних блокувань при обробці даних отримав дані, які вказують на значний ризик виникнення такої ситуації. Програмний засіб, розроблений для цієї функції буде лише виконувати більше функцію спостерігача за роботою процесів, та використанням ресурсів та буде подавати сигнал ядру операційної системи лише тоді, коли виникає значний ризик виникнення ситуації взаємного блокування.

При розробці методу прогнозування ситуації взаємних блокувань потрібно прийняти до уваги тип операційної системи та точну роботу її ядра, також для цього потрібно детально розглянути алгоритм системного виклику. Без оцінки алгоритму планування неможливо розробити конкретний метод, а тим більш, конкретний програмний засіб прогнозування ситуації взаємних блокувань. Тому будуть розглянуть найбільш популярні та найбільш часті у застосування алгоритми планування роботи процесів в операційній системи. Саме планування дає наглядну картину, розстановки пріоритетів процесів, кількість виділеного часу процесам для обробки інформації за допомогою процесора та використання пристроїв введення-виводу і в цілому вказує на

наступний процес, якому буде наданий процесорний час і найбільш активні у використанні ресурсів процеси.

1.6. Висновки

Проведено аналіз виникнення ситуації взаємних блокувань та проаналізовано методи вирішення вказаної проблеми. Всі методи для вирішення даної задачі направлені на упередження або вирішення ситуації взаємних блокувань що виникла. Жоден з них не оцінює вірогідність виникнення даної ситуації, а впливом на роботу планувальника та організацію процесів знижує ефективність роботи операційної системи, тому було прийнято рішення запропонувати новий метод вирішення тупикових ситуацій оцінюючи роботу процесів в операційній системі та всіх задіяних ресурсів існуючими процесами і приймати рішення лише в тому, випадку, коли ризик виникнення ситуації взаємних блокувань значний.

2 РОЗРОБКА МЕТОДУ ПРОГНОЗУВАННЯ СИТУАЦІЇ ВЗАЄМНИХ БЛОКУВАНЬ

2.1. Вибір методу планування

Всього існує три класи планувальників в операційних системах:

- пакетний - орієнтований на тривалі завдання, які вимагають значних обчислювальних ресурсів, де не потрібно часто виконувати переривання. Тобто мається на увазі обробка великих за обсягом завдань великими пакетами і немає обмеження на час виконання.
- інтерактивний - орієнтований на зменшення часу відгуку, тобто щоб складалось враження, що система дуже швидко відповідає на запити. Звичайні призначені для користувача системи на ПК - це інтерактивні системи, коли у відповідь на дію користувача (наприклад переміщення миші) ОС виконує якусь дію. І завжди кожному користувачеві хочеться, щоб ця відповідь відбувався якомога швидше. Головне щоб запит, який було надіслано в систему, отримав максимально швидко відповідь. Запит є будь-якою взаємодією з комп'ютером.
- реального часу - спеціалізований клас, орієнтований на жорстко обмежений час обробки - граничний термін завершення будь-якої роботи. Головне, щоб певним чином процес виконання роботи завершувався до певного терміну, це поняття називається дедлайн. Отриманий запит повинен бути оброблений не більше, ніж в певний проміжок часу. Класичний приклад системи реального часу - управління ядерним реактором, в якому перевищення часу відгуку призведе до аварійної ситуації.

Рівні планування поділяються на три типи: довго часовий, середньо часовий та коротко часовий. Суть довго часового типу заключається в тому, що

перш за все планувальник вирішує які задачі будуть добавлені. Середньо часовий – планувальник вирішує потрібно на певний час вигружати програму в пам’ять нижчого рівня, чи ні. І планувальник коротко часовий вирішує, якому потоку дати наступний квант процесорного часу і якої довжини. Координує виконуються потоки на різних центральних процесорів.

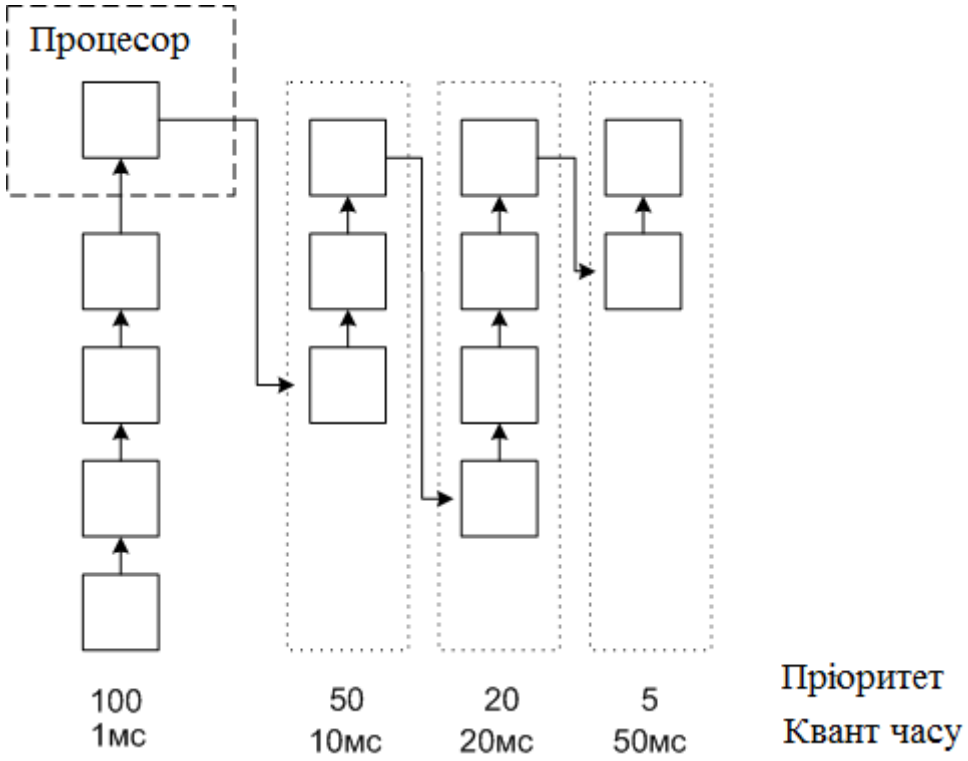

Рисунок 2.1 Процес роботи циклічного планування

Для здійснення поставлених цілей стратегії планування повинні спиратися на будь-які характеристики процесів в системі, завдань в черзі на завантаження, стану самої обчислювальної системи в кожен момент часу, і, загалом, на параметри планування. У цьому розділі буде описано та враховано для розробки методу прогнозування взаємних блокувань ряд таких параметрів, не претендуючи на повноту викладу. Всі параметри планування можна розбити на дві великі групи: статичні параметри і динамічні параметри. До статичних параметрів обчислювальної системи можна віднести граничні значення її ресурсів.

Стратегії довгострокового планування використовують в своїй роботі статичні і динамічні параметри обчислювальної системи і статичні параметри самих процесів (динамічні параметри процесів на етапі завантаження завдань не можуть бути відомі). Стратегії короткострокового і середньострокового планування також враховують і динамічні характеристики всіх працюючих в системі процесів. Для середньострокового планування, в якості таких характеристик, може виступати наступна інформація: скільки часу пройшло з моменту вивантаження процесу на жорсткий диск або його завантаження в оперативну пам'ять, скільки оперативної пам'яті займає сам процес, скільки процесорного часу було вже виділено планувальником процесу.

Для короткострокового планування знадобиться ввести ще два динамічних параметра. Діяльність будь-якого процесу можна представити як послідовне виконання циклів переходу процесу зі стану «роботи» в стан «блокування». Процес планування здійснюється частиною операційної системи, яка називається планувальником. Планувальник може приймати рішення про вибору виконання нового процесу, з числа що знаходяться в стані «готовність», в наступних чотирьох випадках: коли процес переходить зі стану: «роботи» в стан «завершення», зі стану «роботи» в стан «очікування», зі стану «виконання» в стан «готовність», наприклад, після переривання по таймеру, завершилася операція введення-виведення і процес, отримавши потрібні йому для роботи дані, переходить від стану «очікування» в стан «готовність» . Детально процедура такого переведення не буде розглянута, оскільки вказаного для розробки методу прогнозування достатньо, щоб розробити алгоритм методу прогнозування.

У випадках коли процес, який перебував в стані «роботи», не може далі виконуватися, і для виконання завжди необхідно вибрати новий процес. У інших випадках, коли планування може не проводитися, то процес, який виконувався до переривання, може продовжувати своє виконання після обробки переривання. Якщо планування здійснюється тільки у випадках коли процес добровільно звільнив процесор, кажуть, що має місце невитісняючі

планування. В іншому випадку говорять про витісняюче планування. Термін витісняюче планування виник тому, що процес, який виконую свою роботу, проти своєї волі може бути витіснений зі стану " роботи" іншим процесом.

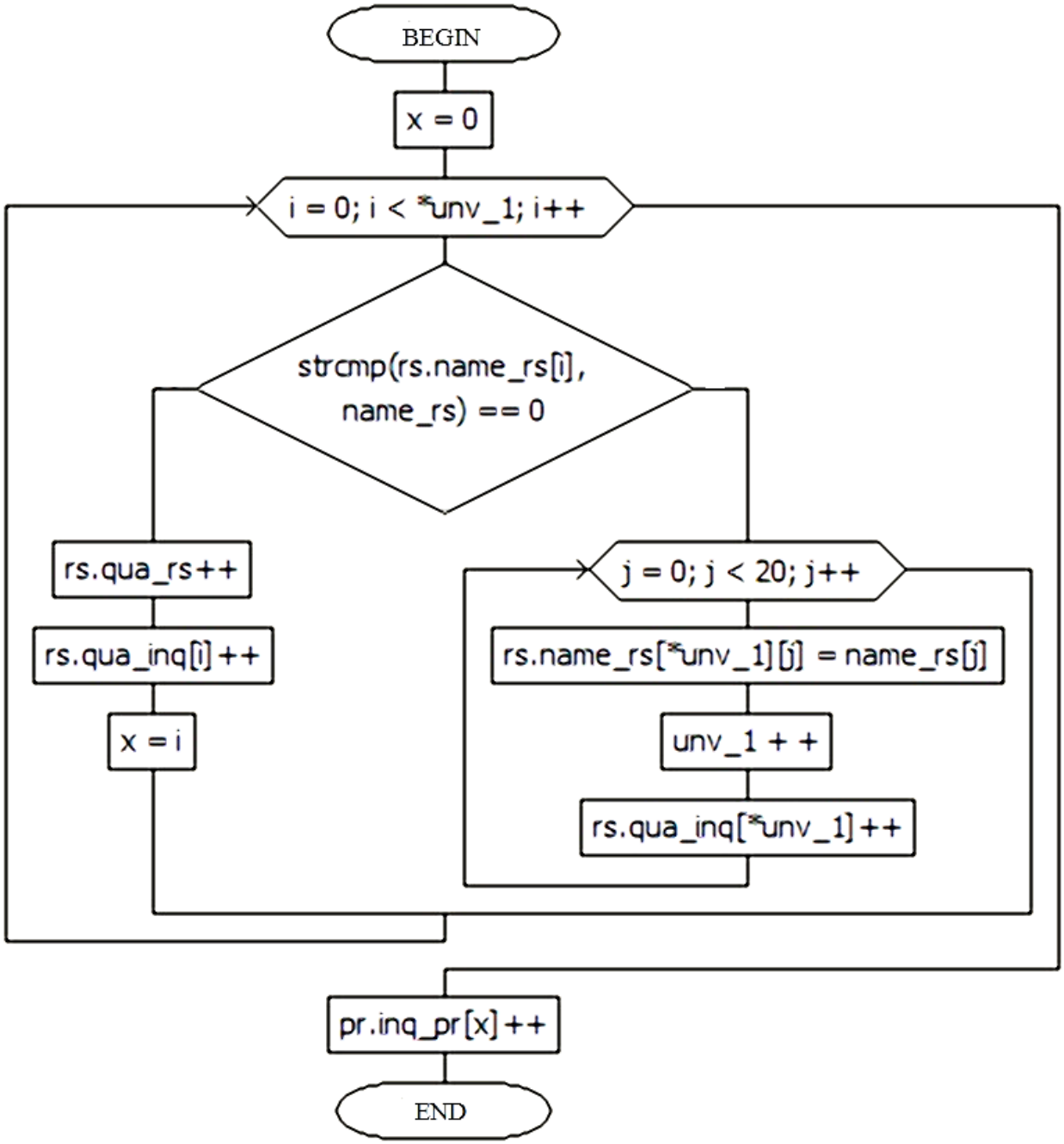

Рисунок 2.2 Алгоритм обробки запитів від процесів до ресурсів

При доступності тільки до одного ядра центрального процесору, необхідно зробити вибір, який з вказаних процесів буде виконуватися наступним. Та частина операційної системи, на яку покладено цей вибір, йменується планувальником, а алгоритм, який вона використовує, носить назву алгоритм планування.

Планувальник поряд з найбільш раціональним вибором процесу для наступного виконання, повинен враховувати також ефективне завантаження центрального процесора, оскільки перемикання процесів є досить клопітним заняттям. Спочатку має відбутися перемикання з режиму інтерфейсу користувача в режим ядра, потім збережено стан поточного процесу, включаючи збереження його регістрів в таблиці процесів для їх подальшої пере завантаження. На деяких системах повинна бути збережена також карта пам'яті (наприклад, ознаки звернення до сторінок пам'яті). Після цього розпочинає свою роботу алгоритм планування для вибору наступного процесу. Потім, відповідно з картою пам'яті нового процесу, потрібно пере завантажити блок управління пам'яттю. І, тільки після виконання вказаних інструкцій, новий процес повинен бути запущений. В додаток, до всього перерахованого, перемикання процесів знецінює весь кеш пам'яті, змушуючи його двічі динамічно завантажуватися з оперативної пам'яті, після входу в режим ядро і після виходу з цього режиму. Через це, занадто часте перемикання може поглинути значну частину процесорного часу, тому частого перемикання між процесами потрібно уникати.

Практично у всіх процесів піки обчислювальної активності чергуються з запитами введення-виведення. Найчастіше центральний процесор деякий час працює без зупинок, потім відбувається системний виклик для зчитування даних з файлу, або для їх запису в файл. Коли системний виклик завершується, центральний процесор відновлює обчислення даних до тих пір, поки йому не знадобляться додаткові дані для продовження розрахунків або не буде потрібно записати додаткові дані в пам’ять. Слід зауважити, що деякі операції введення-виведення вважаються обчисленнями. Коли центральний процесор копіює біти,

що відповідають за відображення даних, в пам'ять, щоб оновити зображення на екрані, то він зайнятий обчисленням, а не введенням-виведенням, оскільки при цьому задіяний він сам. В таких ситуаціях введення-виведення відбувається в тому випадку, коли процес блокується, і переходить в стан очікування, поки зовнішній пристрій завершить свою роботу.

Процеси, що обмежені швидкістю обрахунків – це процеси, що проводять основну частину свого часу за обчисленнями, а процеси, що обмежені роботою пристроїв вводу виводу, навпаки, більшу частину часу свого існування проводять у стані блокування, очікуючи поступу потрібних, для обчислення, даних. Процеси, які обмежені швидкістю обчислень, як правило мають тривалі піки обчислювальної активності і, відповідно, менш часті періоди очікування введення-виведення, а процеси, що обмежені швидкістю роботи пристроїв введення-виведення, мають незначні періоди активності центрального процесора і, відповідно, доволі часто знаходяться у стані блокування, очікуючи поступу нових даних для обчислення. Також ключовим фактором тут є період пікової активності кожного ядра центрального процесора, а не тривалість активності пристроїв введення виводу. Процеси, обмежені швидкістю роботи пристроїв введення-виводу, вважаються такими тільки тому, що не займаються тривалими за часом обчисленнями в проміжках між запитами введення-виведення, а не тому, що вони головним чином зайняті частими запитами введення-виведення. Запит на читання блоку даних з диска займає одне і те ж час незалежно від того, багато чи мало часу йде на їх обробку після отримання.

Якщо апаратний таймер забезпечує періодичні переривання з частотою 50 або 60 Гц або з якоюсь іншої частотою, то планувальник зобов’язаний приймати рішення при кожному перериванні по таймеру або при кожному перериванні. За реакцією на переривання по таймеру, алгоритми планування можна розділити на дві категорії. Непріоритетний алгоритм планування вибирає процес, який повинен запускатися наступним, а потім просто дає йому можливість виконуватися до тих пір, поки він не заблокується в очікувані або завершення операції введення-виведення, або іншого процесу, або до тих пір,

поки він не завершиться свою роботу та добровільно звільнить одне з ядер центрального процесору. Навіть якщо процес буде працювати протягом багатьох годин, його не буде примусово переведено зі стану роботи у стан готовності. В результаті, під час переривань за таймером не буде прийнятих рішень. Після завершення обробки переривання по таймеру, свою роботу відновить той процес, що раніше запускався, якщо тільки який-небудь процес більш високого рівня не буде перебувати у стані очікування.

На відміну від цього пріоритетний алгоритм планування передбачає вибір конкретного процесу і надання йому можливості працювати до закінчення деякого строго відведеного періоду часу. Якщо до закінчення цього періоду він все ще буде працювати, планувальник призупиняє його виконання і вибирає для запуску іншого процес, якщо є хоча б ще один процес в системі у стані готовності. Здійснення пріоритетного алгоритму планування вимагає наявності переривань за таймером, що виникають коли закінчується певний період часу, щоб повернути управління одному з ядер центрального процесору планувальником. Якщо переривання по таймеру недоступні, залишається лише використовувати непріоритетні методи планування.

Якщо центральний процесор і всі пристрої введення виводу мають змогу бути постійно задіяні, то буде зроблено значно більший обсяг обрахунків в секунду. В пакетної системі, планувальник управляє тільки тим завданням, яке є в наявності в пам'яті для виконання. Одночасне розміщення в пам'яті частини процесів, що обмежені швидкістю обчислень, і частини процесів, що обмежені швидкістю роботи пристроїв введення виводу, буде більш доцільним рішенням, ніж завантаження і виконання спочатку абсолютно всіх завдань, що обмежені швидкістю обчислень, а потім, коли їх виконання вже завершиться, завантажуються і виконується всі завдання, обмежених швидкістю роботи пристроїв введення виводу. Якщо використовується остання з вказаних стратегій, то при запуску роботи процесів, обмежених за швидкістю обчислень, вони будуть конкурувати за користування центральним процесорів, а пам’ять пристрій буде простоювати. Потім, коли потрібно буде виконувати обчислення,

то процеси, які обмежені за швидкістю роботи пристроїв введення виводу, вступлять в боротьбу за накопичувач на комп'ютері.

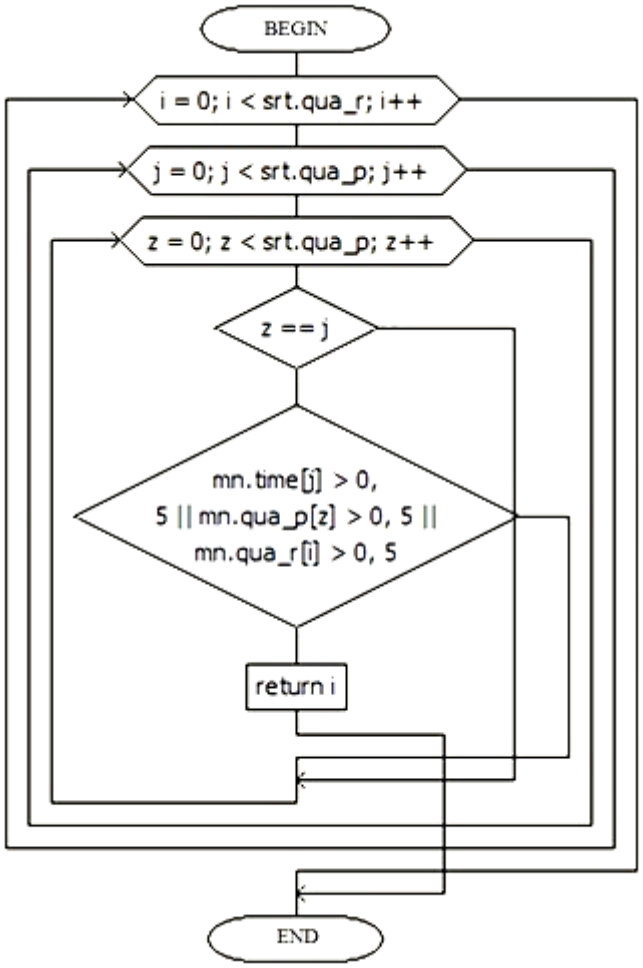

Рисунок 2.3 Алгоритм обробки стану процесів

2.2. Розробка методу прогнозування взаємного блокування з урахування циклічного планування

Одним найпростіших і найбільш часто використовуваних вважається алгоритм циклічного планування. Кожному процесу призначається певний інтервал часу, який йменується квантом, в проміжок вказаного часу надається можливість виконання процесу. Якщо процес до завершення кванту часу ще виконується, то ресурс одного з ядер центрального процесора у нього відбирається і передається іншому, який знаходиться наступним в черзі,

процесу. Якщо процес переходить в заблокований стан або завершує свою роботу до закінчення кванта часу, який йому виділив планувальник, то перемикання центрального процесора на інший процес відбувається саме в цей момент. Алгоритм циклічного планування не представляє складності в реалізації. На рис. 2.4, показано, що від планувальника потрібно всього лиш вести список процесів, готових до виконання за своїм станом. Коли процес вичерпає виділений йому квант часу, він, як показано на рис. 2.4, поміщається в кінець списку черги.

Рисунок 2.4 Циклічне планування

Єдине, що по-справжньому цікавить в циклічному плануванні, - це тривалість кванта часу, що і буде враховано в розробці методу прогнозування взаємних блокувань. Перемикання з одного процесу на інший вимагає певної кількості часу для виконання завдань обслуговування роботи процесів - збереження і завантаження регістрів і карт пам'яті, оновлення таблиці процесів і списків, запису на диск і перезавантаження кешу пам'яті і т. п. Припустимо, що перемикання процесу, або перемикання контексту, як його ще іноді називають, займає 1 мс, включаючи перемикання карт пам'яті, запис на жорсткий диск і перезавантаження кеша і т. п. Також, важливо врахувати, що значення кванта часу встановлено на 4 мс. При таких параметрах настройки після 4 мс корисної роботи центрального процесора доведеться затратити (тобто втратити) 1 мс на перемикання процесу. Таким чином, п’ята частина процесорного часу буде витрачено на реалізації перемикання між процесами, а це, поза всяким сумнівом, дуже багато.

Щоб підвищити ефективність використання та користь від центрального процесора, ми можемо встановити значення кванта часу рівним 100 мс. Тепер

втрачається всього 1% часу. Але на серверній системі, якщо за дуже короткий час до неї надійде 50 запитів, що мають значний ступень потребі у центрального процесора. У список, готових до виконання процесів, буде поміщено 50 процесів. Якщо центральний процесор простоює свій час, перший з них буде запущено негайно, другий не зможе запуститися, поки не закінчаться 100 мс, і всі інші, що стоять за ним в черзі. Якщо всі процеси повністю скористаються своїми квантами часу, то найменш пріоритетний процес, який є останнім в черзі, буде перебувати в очікуванні протягом 5 с, перш ніж отримає можливість запуститись. Багатьом користувачам робота системи при п'ятисекундний очікуванні відповіді на запит, що передбачає коротку команду, здасться дуже повільної. Ця ситуація отримає особливо негативне забарвлення, якщо деякі із запитів, розміщені ближче до кінця черги, вимагають лише кілька мілісекунд процесорного часу. Якщо квант часу буде коротше, якість їх обслуговування покращиться.

Інша особливість полягає в тому, що якщо значення кванта часу встановлено більшим, ніж середній час використання центрального процесора, переходу стану процесу не буде відбуватися занадто часто. Замість цього більшість процесів виконуватимуть операцію блокування перед закінченням виділеного ним кванта часу, що викликає перемикання процесів. Виняток примусового переривання за таймером підвищує продуктивність роботи системи, оскільки перемикання процесів відбувається тільки при логічної необхідності, тобто коли процес блокується і не може продовжити роботу.

Для оцінки прогнозу взаємного блокування дуже важливе розуміння планувальника в операційній системі. Оскільки кожна операційна система має свій алгоритм планування, то вибірка частоти роботи процесів та вибірка надання можливості використання процесора тим, чи іншим процесом, а також розстановка пріоритетів проходять в системі кожного типу по різному. Алгоритм планування є дуже важливим у прогнозуванні взаємних блокувань, тому для кожної операційної системи будуть використовуватися різні програмні засоби задля прогнозування взаємних блокувань.

В даному проекті за приклад береться циклічний алгоритм планування у інтерактивних системах. Він є одним із найпоширеніших так використовуються у багатьох Unix-подібних операційних системах.

На рисунку 2.2 зображено поетапне виконання системного виклику на якому зображено, що першим кроком є звернення користувацької програми до бібліотеки системного виклику. Другим кроком є звернення до диспетчера переривань, цей крок також є кроком переходу в режим ядра операційної системи. Далі диспетчер звертається до обробника системного виклику, який визначає саме який системний виклик обробляється та зчитує код файлу, який відповідає за вказаний користуватською програмою конкретний системний виклик. Після обробки системного виклику робота ядра закінчується і

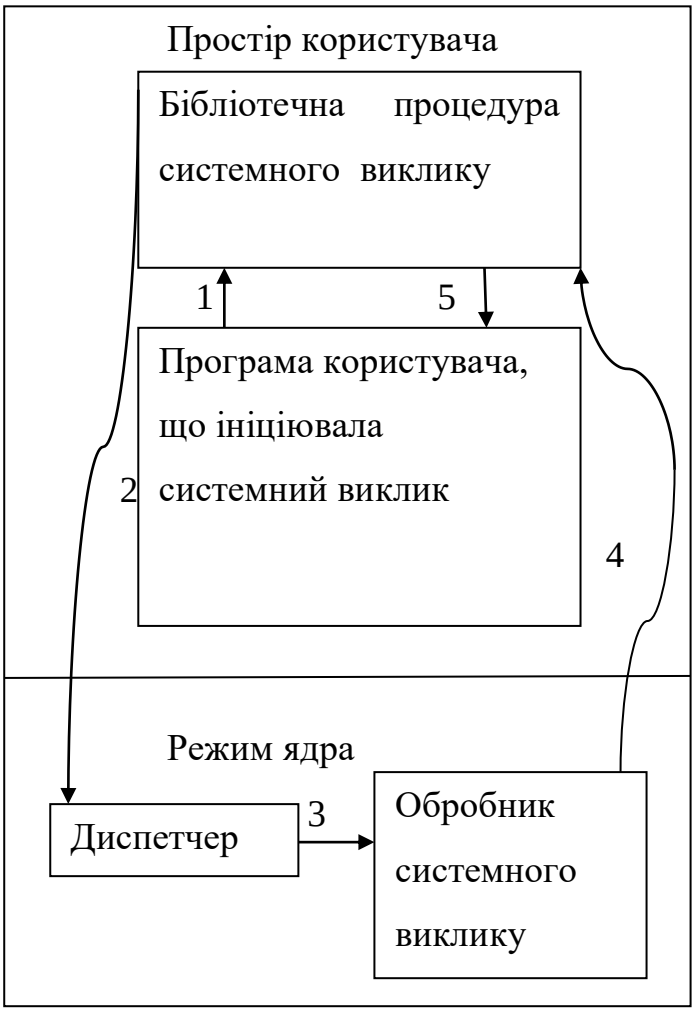

Рисунок 2.5 Поетапне виконання системного виклику

управління передається бібліотечній процедурі, яка і на запит програми користувача і здійснила системний виклик. І останнім кроком є передача управління самій програмі, що здійснила системний виклик.

Оскільки циклічний алгоритм планування передбачає виділення процесам кванту часу для використання роботи процесора, то після вичерпання свого кванту, процес в залежності від його пріоритету, планувальник конкретний процес переміщує вниз черги. Тому процеси з рівним пріоритетом, переміщені в чергу, яка постійно змінюється, мають фактично однакові шанси отримати можливість знову використовувати процесор у своїх цілях. Тому має важливе значення, скільки всього часу, на протязі свого існування, конкретний процес перебував у стані роботи. Оскільки циклічне планування надає перевагу тим процесам, що використали більше свого квантного часу аніж інші, мають менше шансів отримати у своє розпорядження процесор та порти вводу та виводу. А при прогнозуванні взаємних блокувань є важливим знання того, кому наступного разу буде надана можливість перейти зі стану готовності до стану роботи.

Також є дуже важливий фактор перебування процеса у стані готовності, оскільки кількість, часу проведеного у даному стані вказує на активність процеса про роботі з даними та його потребу в процесорі за для обробки даних, якими він оперує. Перебування в стані готовності, вказує нате, що процес зазвичай, знаходиться у стані очікування, коли поступлять конкретні дані і лише тоді, даним процесом буде подано запит на використання процесору або пристроїв вводу виводу.

Оскільки процеси конкурують між собою за процесорний час та ресурси заради виконання ними певних функцій, то можна зробити висновок, що процес, який зазвичай знаходиться у станах готовності та роботи, є активним у роботі та ймовірність, щ саме такого типу процес опиниться у ситуації взаємного блокування надзвичайно висока. А процес, який навпаки, знаходиться у стані блокування та постійного очікування вхідних даних, не

часто подає запит на використання процесу, тому ймовірність того, що він опиниться у стані взаємного блокування значно менша.

Також, дуже важливим є оцінка кількості ресурсів, та запитів на їх використання тим, чи іншим процесом. Якщо ресурс отримує багато запитів від багатьох процесів на передачу та використання його даних, то є значна вірогідність того, що саме цей ресурс може бути причиною взаємного блокування між процесами.

Тому дуже важлива оцінка кількості всього запитів до всіх задіяних ресурсів у системі, також важливо мати дані, щодо кількості запитів використання конкретного ресурсу процесами. Це дасть можливість визначити можливу причину майбутнього взаємного блокування та запобігти їй. Важливо оцінити кількість запитів конкретного процеса до конкретного ресурса, якщо часто звернення до конкретного ресурса є в двох і більше процесів, то ймовірність взаємного блокування між ними є значною.

При написані програмного засобу для обрахунків, буде враховуватись кількість запитів на використання конкретного ресурсу конкретним процесом і оцінки кількості таких запитів від різних процесів, тому що така ситуація значно підвищує можливість взаємного блокування. Також будуть враховані часи роботи та стан готовності кожного процесу, відносно всього часу існування конкретного процеса. Стан блокування процесу враховуватись не буде, тому що це вказую на його малу активність. Дуже важливо врахувати кількість запитів до кожного задіяного ресурса у системі відносно всієї кількості запитів, це вкажи на ресурси, до яких найчастіше звертаються за даними для обробки процеси, що і може стати причиною виникнення взаємного блокування.

Неможна не звернути увагу на кількість процесів, що найчастіше використовують дані того чи іншого ресурса. Тому відносно кожного задіяного у системі ресурсу, буде оцінюватись кількість звернених до нього процесів, відносно всього кількості існуючих процесів. Це вкаже на те, як часто процеси звертаються до конкретного ресурсу.

2.3. Розробка методу прогнозування взаємного блокування з урахування конкуренції процесів за використання ресурсів

При роботі операційної системи всі процеси конкурують між собою не тільки за право використання процесорного часу, а і за всі ресурси з метою виконання поставленої для них задачі. Зазвичай процеси конкурують між собою не тільки за право використання пристроїв вводу виводу, але і за використання простих файлів конкуренція між процесами не є рідкістю, тому потрібно враховувати абсолютно всі файли, до яких, під час виконання поставлених перед ними задач, процеси подають запит на виконання.

Дуже важливо приділити увагу тому що є вигружені та невигружені ресурси. Ресурси вигруженого типу можна відібрати у процеса під час виконання і це не приведе до серйозних збоїв у роботі процеса. До ресурсів такого типу зазвичай відносяться файли в пам’яті, що містять дані. Ресурси не вигруженого типу відібрати у процеса значно важче, ніж ресурси вигруженого типу, це пов’язано з тим, що коли у процеса відбирають право на використання даного ресурсу в саме той час, коли процес проводить з ним обмін інформації, це може призвести не тільки до серйозного пошкодження в роботі самого процеса, а й до зависання самої операційної системи та самого ресурсу. Ресурсами такого типу є файли які відповідають за зв’язок з пристроями введення та виводу.

Тому при розробці методу прогнозування взаємних блокувань дуже важливо врахувати вище вказаний фактор. Коли буде надходити значний запит до не вигруженого ресурсу, то ризик того, що виникне ситуація взаємного блокування в порівнянні з вигруженим ресурсом не зміниться, але те, що в конкретного процеса буде непросто відібрати право на використання на займаний ним ресурс без шкоди для роботи операційної системи потрібно врахувати. Тому коли буде надходити запит від різних процесів, що вже зайняли інші ресурси, на використання ресурсів не вигружаємого типу,

обов’язково програмний засіб буде про це інформувати планувальник і останньому потрібно буде прийняти більш кардинальне рішення до конкретних процесів, що йому було вказано, та більш швидко реагувати.

Припускається, що при роботі системи, при кожному системному виклику на запит конкретного ресурсу від певного процеса, інформація про даний ресурс буде заноситись програмним забезпеченням в базу ресурсів. При цьому не будуть враховуватися всі ресурси, яких десятки тисяч в операційній системі, а лише ті, що задіяні існуючими процесами під час виконання інструкцій.

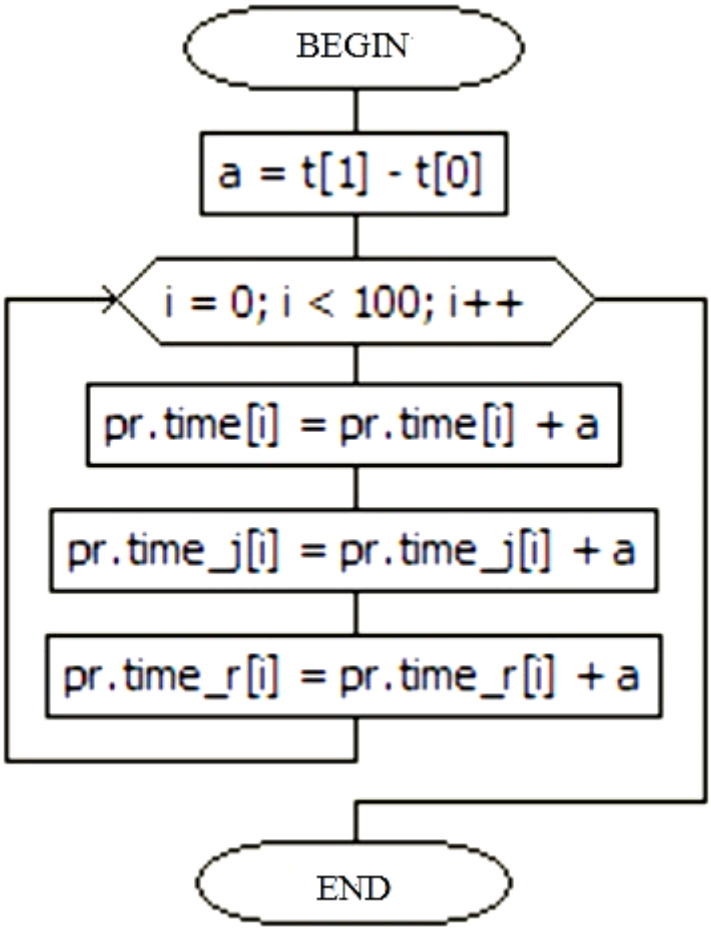

Рисунок 2.6 Алгоритм обробки часу для кожного процесу

Також дуже важливий для розуміння один факт, що при розробці методу прогнозування не має значення чи було задовільнено запит процеса на

використання того, чи іншого ресурсу. Важливо врахувати лише те, що сам запит був і це, в свою чергу, вказує на потребу даного ресурса в різних процесів. Тому при передачі даних від системного виклику, не будуть передаватись ніякі дані про подальше рішення ядра, щодо передачі управління процесу чи задовільнення його запиту на використання вказаного ним ресурсу. Оскільки це тільки б ускладнило програму і зробило б її більш об'ємною, і не має ніякого практичного значення для розробляє мого методу, то всі дані, які потрібні для прогнозування будуть передаватись на початку виконання системного виклику в ядрі.

Тому базу ресурсів можна прийняти як за множину,а кожен ресурс є елементом цієї множини. Кількість запитів на використання для кожного ресурсу є числовим значення для кожного елементу множини, чим більше це значення, тим більше ризик виникнення ситуації взаємного блокування саме через цього ресурсу, тому для того, щоб врахувати всі ресурси, які найчастіше використовуються процесами потрібна одна зміна, яка рахує всі запити на використання ресурсами абсолютно від всіх процесів. Сигнал планувальнику буде посилатися тоді, коли до найбільш часто використовуваних ресурсів звертаються декілька процесів, які в свою чергу , займають інші ресурси, і чим більше було надіслано запитів на використання займаних ресурсів процесами, тим більше буде підвищуватись вірогідність виникнення ситуації взаємного блокування.

2.4. Розробка методу прогнозування взаємного блокування з урахування конкуренції між процесами

Неможливе прогнозування ситуації взаємних блокувань без урахування даних про всі запущені в операційній системі процеси. Тому щоразу коли буде відбуватися системний виклик, в програмне забезпечення будуть передаватися дані про процес щодо його роботи. Таким чином необхідно створити базу даних про процеси, яка звичайно, буде значно відрізнятися від таблиці процесів, яка

міститься в ядрі операційної системи, простотою, оскільки заради прогнозування взаємних блокувань потрібно набагато менше інформації про процеси ніж та, яка міститься в її таблиці.

Найголовнішим даним в базі даних про процеси буде назва всіх існуючих процесів. Щоразу, коли буде виконуватись системний виклик, який передбачає створення нового процесу, то ці дані про новий процес будуть передаватись в базу даних процесів. Також, важливо відмітити, що при розробці методу прогнозування немає ніякої потреби урахування ієрархії процесів, оскільки ймовірність виникнення ситуації взаємних блокувань між материнськими процесами, та тими процесами, що вони створили, не зменшується, тобто така сама як і між процесами, які ніяк не пов’язані з ієрархією.

Дуже важливо відмітити різниці маж процесами, що залежать від використання процесорного часу та процесами, що більш залежні від пристроїв введення та виводу, якщо процес частіше подає запити на використання ресурсів, то саме його дії найбільш вірогідніше приведуть до ситуації взаємних блокувань. Проте процеси, що конкурують за використань процесорного часу, також є небезпечними для взаємних блокувань, оскільки, якщо такі процеси потраплять у таку ситуацію, то вони будуть викликати простоювання процесору та не використовувати його у своїх цілях, очікуючи отримання дозволу на використання вже займаного іншим процесом ресурсу. Найбільш менша вірогідність виникнення взаємного блокування є для фонових процесів, які за частую просто очікують певних подій, які призведуть до поступу до них інформації, та знаходяться найбільшу частину свого часу в заблокованому стані.

Тому для кожного процеса, буде враховуватись кількість запитів на використання ресурсу, які ним були подані за допомогою певного системного виклика. Чим більше процес подав запитів на використання ресурсів, тим більша ймовірність виникнення взаємного блокування у якому буде задіяний саме цей процес. З цього можна зробити висновок, що кожен процес в базі даних про процеси буде мати власний лічильник кількості запитів, що були

викликані ним, заради використання ресурсу. Процеси з найбільшою кількістю запитів на використання ресурсів, бідіть розглядатися як найбільш небезпечні для виникнення ситуації взаємних блокувань. Для того, щоб визначити такі процеси, потрібно мати загальний лічилок всіх запитів на використання ресурсів, який було вказано в попередньому розділі.

Для вирішення проблеми вірогідності взаємного блокування між процесами, що залежні від процесорного часу потрібно враховувати час, який процеси провели у стані готовності та роботи. Також важлива оцінка пріоритету процеса, для передбачення ймовірності передачі йому переваги планувальником. Проведення часу в стані блокування обернено пропорційне часу проведеного в стані готовності та роботи. Саме готовність процеса використовувати у своїх цілях процесор вказує на його процесорну залежність. Тому при розрахунку суми всього часу, проведеного процесом у стані роботи, та часу, проведеного процесом у стані готовності, і поділу цієї суми на весь час існування процесу вкаже на процесорну залежність процеса, оскільки чим більше число і чим частіше воно приближається до одиниці, тим більш процесорнозалежний даний процес.

Тому база даних процесів буде більш об’ємною, ніж база даних ресурсів і також буде розглядатись як множина. Кожен процес буде розглядатися як окремий елемент цієї бази, при цьому матиме параметри кількості запитів на використання ресурсів, кількості часу проведеного у стані готовності, кількості часу, проведеного у стані готовності та весь час існування.

2.5. Застосування теорії множин в прогнозуванні ситуації взаємних блокувань

Для того щоб вивести кінцевий зв’язок між базами даних процесів і ресурсів, потрібно обрахувати відношення кількості запитів до конкретного ресурсу до кількості запитів всього в операційній системі. Чим більше число наближене до одиниці тим більше потреби викликає даний ресурс в існуючих

процесів. Таке саме відношення потрібно обрахувати для процесів в операційній системі, чим більше запитів на використання ресурсів подає кожен процес, тим більша вірогідність того, що саме цей процес потрапить у ситуацію взаємних блокувань.

Тому потрібно створити окрему базу, для обрахунків відношень кількості запитів до ресурсу, кількості відношень кількості запитів поданих процесом і обрахунків кількості часу проведеного у стані готовності та роботи. Далі потрібно обрахувати композицію відношень між вказаними обрахованими даними. Враховуючи що саме ресурси є причиною конкуренції процесів, то в композиції відношень буде використовуватись попадання відношення часу проведення в вказаних стані процесів до відношення кількості запитів відносно кількості всього запитів на використання ресурсів. Між цими двома відношеннями буде вказуватись відношення кожного ресурсу кількості запитів до нього відносно всього кількості запитів на використання ресурсів.

Рисунок 2.7 Стани роботи процесів

При подачі сигналу, що взаємне блокування під значним ризиком. Програма повинна подати сигнал планувальнику і він повинен конкретним процесам знизити пріоритет та відмінити їх запит на вказаний ресурс, а при можливості змусити їх звільнити займані ними ресурси, і, навпаки, іншим процесам віддати перевагу при використані ресурсу та підвищити пріоритет.

2.6. Висновки

В даному розділі проаналізовано роботу планувальника в операційній системі та, найбільш поширений, алгоритм циклічного планування. Також було розглянуто роботу процесів в UNIX-подібних операційних системах. З організації роботи операційної системи було виведено залежність процесів від ресурсі пам’яті, процесорного часу та організації планування. Запропоновано прогнозування на базі кількісного звернення від процесів, що найчастіше знаходяться у стані роботи, до ресурсів до яких най часті надходять запити на використання.

Виведено залежності виникнення ризику взаємного блокування між процесами, що залежні від процесорного часу та пристроїв вводу-виводу, та найчастіше використовувані ресурсами всіма процесами в операційній систем. Виведено, що ризик взаємного блокування підвищується прямо пропорційно при зверненні від процесів, що значну кількість свого часу перебували у стані роботи в системі, до ресурсів, до яких найбільше було надіслано запитів на використання від всіх процесів, що на в конкретний момент часу існують в операційній системі.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МЕТОДУ ПРОГНОЗУВАННЯ ВЗАЄМНИХ БЛОКУВАНЬ

3.1. Написання структур для зберігання даних процесів та ресурсів

Для написання програмного забезпечення було вибрано мову програмування С, оскільки припускається, що дані програмні засоби будуть працювати в ядрі операційної системи. Ідея впровадити програму в ядро системи виправдовується тим, що за всіма процесами та їх показниками потрібен повний контроль та регулярне отримання даних про процеси. Оскільки таку детальну інформацію може лише отримати ядро операційної системи, було прийнято рішення впровадити програмний засіб безпосередньо в ядро операційної системи. Також потрібен повний контроль за всіма ресурсами, що були задіяні процесами. Тому для контролю всіма запитами до ресурсів від кожного з процесів, потрібне саме впровадження програми в ядро. Оскільки практично всі операційні системи в своїй більшості написані на мові програмування С, то було вибрано безпосередньо ця мова програмування.

Для розробленого програмного засобу була вибране ядро UNIX-подібної операційної системи, а саме Linux. Причиною вибору сааме цього ядра було його безкоштовність та доступність, поширеність у використані як і серед звичаних користувачів, так і в спеціалізованому використані.

Також важливо скоординувати програму з файлами, що відповідають за системні виклики, тому для коректної та повноцінної роботи програмного засобу потрібно внести незначні, але дуже важливі зміни в декілька файлів системних викликів, а саме:

- системний виклик fork(), вказаний системний виклик відповідає за створення нових процесів в операційній системі, при виконані даного системного виклику через потік вводу в файл розробленої програми

передаються такі параметри: ім’я нового створеного процесу, його пріоритет та стан в якому запустився щойно створений процес

- системний виклик exit(), що вказує на завершення роботи процесу та перехід його у стан блокування. В цьому випадку передаються параметри часу, який провів процес у стані роботи, використовуючи процесор та параметри зміни стану в стан блокування
- системний виклик kill(), що вказує на завершення роботи процеса, у разі такого системного виклику в параметрах передається лише назва процесу і програма видаляє вказаний процес зі структури процесів та всі його дані
- системні виклики open(), write(), read(), вказані системні виклики повідомляють про звернення певного процесу до конкретного ресурсу, для програми немає значення на що саме було відправлено запит і чи був запит процеса задовільнено, головне, що запит звернення до ресурсу був, що вказує на потребу вказаного процесу у ресурсі, що є першопричиною виникнення конкуренції, яка, в свою чергу, призводить до виникнення ситуації взаємних блокувань. Дані системні виклики передають в параметрах назву процесу, ресурсу, після цього програма збільшує на одиницю кількість звернень до вказаного ресурса, кількості всього звернень та кількості запитів від процеса
- системний виклик setsid(), вказує на передачу правом використовувати можливості процесора іншим процесом. В разі такого системного викликав програму передаються такі дані: ім’я процеса, в якого відібрали право використовувати центральний процесор, та процес, якому передали управління. Також змінюються дані про стан обох процесів та їх час перебування у станах.

Вказані системні виклики найбільш цікавлять розроблену програму, саме від них програмний засіб отримує дані, за допомогою обробки яких програма визначає прогноз виникнення ризику ситуації взаємного блокування між конкретними процесами та ресурси, які можуть стати причиною виникнення

вказаної ситуації. Для цього на мові С створюються дві структури, перша – для занесення даних про ресурси, друга – для занесення даних про процеси.

В структурі до якої заносяться дані процесів створюються три одномірні масиви, а саме масив часу для кожного процеса, масив часу проведеного у стані готовності для кожного процеса і масив стану роботи для кожного процеса. Перший вказаний масив зберігає весь час існування для кожного процеса, другий масив зберігає дані про час проведений у стані готовності роботи з процесором для кожного процеса і третій вказаний масив зберігає дані щодо кожного процеса проведення у стані роботи.

Створюється двомірний символьний масив в який заноситься ім’я кожного процеса, який існує в даний час в операційній системі. Також створюється масив цілих чисел, в який заносяться дані кількості запитів на використання файлів для кожного процеса. Також вводиться додаткова зміна, яка зберігає номер конкретного процесу, і на цю зміну вказує глобальний вказівник. Суть цієї зміної заключається в тому, що коли виникає системний виклик та заносить дані для кожного процесу, він передає ім’я цього процесу, окрема функція оброки знаходить цей процес під порядковим номером у масиві імен процесів, та змінює вказані системним викликом параметри, саме через глобальний вказівник на цю зміну, яка зберігає номер процеса, до якого потрібно внести зміни, вносяться зміни до даних які співпадають з номером процеса у символьному масиву спуску процесів. І останнім масивом цілих чисел є масив діючих станів для кожного процеса в даний момент, всього їх три: готовність, робота та блокування. Якщо для конкретного процеса в цьому масиві записана 1, то він знаходиться у стані роботи з процесором, якщо 0 – то це стан блокування і процес очікує дані для обробка, якщо 2 – то це стан готовності і процес у черзі, яку вистроїв планувальник.

Структура в якій зберігаються дані щодо кожного ресурсу, який був задіяний тим, чи іншим процесом, містить в собі, перш за все, двомірний масив в який заноситься ім’я кожного ресурсу , до якого було хочу б одно звернення від любого процесу. Також там міститься зміна, яка рахує абсолютно всі

звернення від всіх процесів до всіх ресурсів. Також створюється масив цілих чисел, який містить кількість звернень до кожного ресурсу від любого процеса, що вказую на потребу того, чи іншого ресурса серед процесів. Як і в структура процесів, в базі даних ресурсів присутня зміна, яка вказує на конкретний ресурс, на яку, в свою чергу, вказую глобальний показник. Коли під час системного виклику вносяться зміни до якогось конкретного ресурсу, саме через функцію обробки ресурсів глобальному показнику передається номер ресурсу в символьному списку ресурсів, до якого саме вносяться зміни.

Саме до цих двох структур вносяться дані для подальшої обробки з метою здійснення прогнозу взаємних блокувань. Важливо звернути увагу на те, що введення лише однієї зміної, яка вказує на конкретний процес, чи ресурс, значно економить пам’ять і не потрібно вводити ще 2 масиви цілих чисел, і можна лише за допомогою вказівників коректно обробляти інформацію щодо кожного процеса чи ресурса.

3.2. Написання функцій для обробки даних структур процесів та ресурсів

Для того щоб всі дані, які передаються системними викликами коректно заносились у структури та оброблялись потрібно ввести для цього ряд таких функцій: функція створення нового процесу, функція знищення процесу, функція зміни стану процесу, функція часу для кожного процесу, функція обробки даних від системних викликів звернення процеса до ресурсу.

Функція створення нового процесу отримає дані про ім’я нового процесу, та записує його символьний масив процесів. Зміна, що відповідає за кількість процесів в операційній системі інкрементується, а сам процес заноситься в самий кінець двомірного символьного масиву. Також масив, що відповідають за час процеса проведений в тому чи іншому стані і масив кількості звернення процеса до ресурсів отримують значення 0. І масиву, що відповідає за стан процеса, передається параметр який було передано системним викликом.

Функція знищення процесу отримує лише один параметр – це ім’я процесу, який потрібно знищити. Після того, яки їй було передано відповідний параметр, функція запускає цикл пошуку ім’я процеса в двомірному символьному списку процесів і після перебору знаходить його номер у списку та присвоює нуль всім даним, що знаходились в інших масивах. Для того, щоб зберегти відповідність між іменами процесів та даними про їх стан, у всіх масивах, починаючи від видаленого процеса, кожному номеру в списку присвоюється попереднє значення.

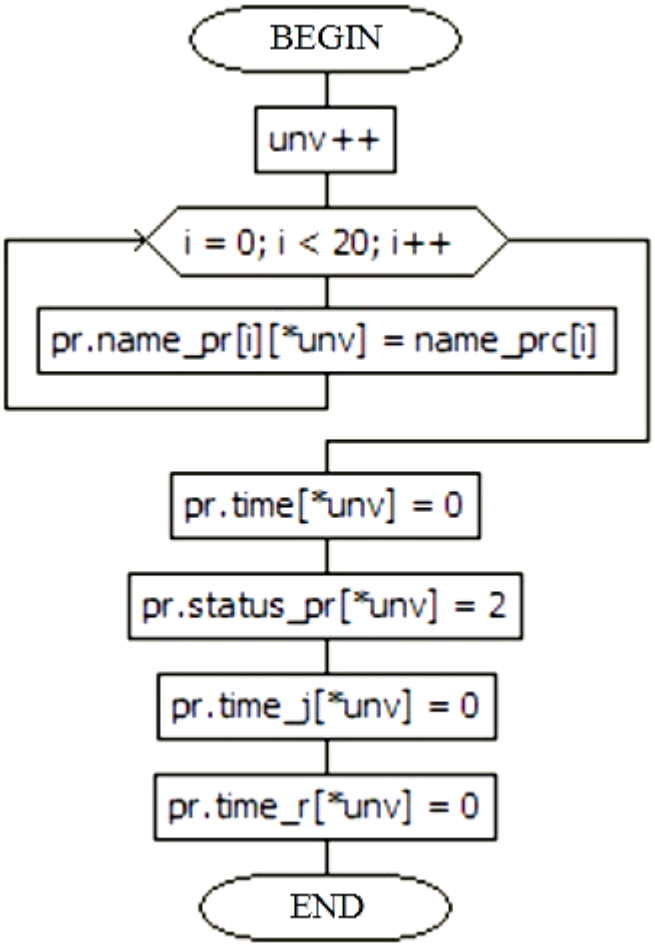

Рисунок 3.1 Алгоритм обробки завершення процесу

Системній виклики читання, виконання та запису у файли для розробляю чого методу не мають суттєвої різниці, тому для них розроблено однакову функцію. Функція обробки даних від системних викликів звернення процеса до

ресурсу запускається лише в тому випадку, коли відбувся системний виклик звернення до ресурсу від процеса, тобто запит на читання файлу, на виконання файлу, на запису даних у файл. У даній функції запускаються два цикли, перший з яких звертається до структури ресурсів і знаходить ресурс до якого, було проведено звернення, а другий цикл звертається до масиву процесів, який подав запит на використання вказаного ресурсу. Коли вказані ресурс та процес були знайдені, то їх значення кількості звернень інкрементується. Також інкрементується зміна, яка відповідає за кількість звернень від процесів до ресурсів всього.

Всі описані функції є необхідними для того, щоб обробляти всі дані, що поступають до програми та контролювати відповідність між двомірними символьними масивами та даними що відповідають і процесам та ресурсам. Також введення глобальних вказівників значно зекономило пам’ять від створення ще двох масивів цілих чисел.

3.3. Написання функцій для розрахунку композиції відношень

Для розрахунку даних в структурі процесів та структурі ресурсів в програмі створюються такі функції: функція обробки процесного часу, функція обробки запитів на використання ресурсів від процесів, функція обробки кількості запитів від одного процеса, функція обробки кількості запитів до ресурсів, функція композиції відношень між всіма вищевказаними функціями. Для збереження всіх даних потрібно в програму ввести додаткову структуру, що зберігатиме проміжні дані. Перші дві вказані функції відносяться до процесної взаємодії та їх кількісно звернення до всіх задіяних в роботі ресурсів. Лише одна функція відноситься до ресурсів, вона дає інформацію про те, до якого ресурсу найбільше було надіслано запитів у відсотковому відношенні. Остання вказана функція обробляє дані попередніх трьох функцій та показує композицію відношень між функціями обробки процесів через дані функції ресурсів.

Функція обробки кількості запитів на від одного процесу розраховує в відсоткових даних кількості звернень від одного процеса до всіх ресурсів, яких ним було передано запит на їх використання, до абсолютно всіх запитів від всіх існуючих процесів в операційній системі. При обробці таких даних вносяться до масиву з плаваючою точкою, який заносить в новостворену структуру. Кількість запитів на використання різних ресурсів від кожного процесу діляться на кількість всього запитів звернення до всіх задіяних ресурсів. Така проста функція обробки вказує на процеси, які найбільш часто подавали запит на дозвіл використання ресурсів. Саме такі процеси найчастіше є учасниками виникнення ситуації взаємних блокувань.

Функція обробки процесного часу вказує на найбільш пріоритетні процеси і операційній системі та на ті процеси, які зазвичай конкурують між собою на право використовувати процесорний час. Всього процес може перебувати у трьох станах, це стан блокування, коли процес не подає ніякого запиту на опрацювання, навіть якщо процесора немає ніякої іншої роботи, це стан готовності, це коли процес подав запит на роботу, але центральний процесор зайнятий іншим процесом, тому в залежності від пріоритету та використаного кванту часу, планувальник задає процесові певне місце в черзі, та стан роботи, це коли процес використовує в цей момент часу процесор. За допомогою суми часу перебування процеса у стані готовності та стану роботу, та їх поділу на весь час існування процеса в операційній системі отримується результат який вказує на активність процеса. Чим менше отримане значення, тим активніше процес, та більшість свого часу подавав запити на використання центральний процесор, і , навпаки, чим більш результат наближено до одиниці, тим більше часу процес перебував у заблокованому стані. Виключенням є значення один, якщо отримано саме таке значення, то процес весь час свого існування перебував або у стані готовності, або у стані роботи, але ніколи у стані блокування. Цей результат вказує на найбільш активні процеси, конкуренція яких є найбільшою причиною виникнення ситуації взаємних

блокувань. Дані для кожного процеса заносяться у створений масив, який зберігається у структурі проміжних даних.

Функція обробки ресурсних даних оперує всього лиш двома змінними, це зміні кількості всього звернень до ресурсу за весь час роботи операційної системи, та зміна кількості звернень до кожного ресурса. Перша зміна для всіх однакова, як було вказано вона змінюється при кожному звернені до будь-якого ресурсу від будь-якого процеса. А ось зміна кількості звернень до ресурсу в кожного ресурсу своя і записується в масиві, який зберігається в структурі ресурсів. Тому таким чином, за допомогою цикла перебирається вищевказаний масив та поділом значення кількості звернень до кожного ресурсу на кількості звернень до ресурсів всього, вказує на найбільш запитує мий в операційній системі ресурс, чим більш значення наближене до одиниці тим більше запитів було надіслано до цього ресурсу відносно всіх задіяних в операційній системі ресурсів. Тут також може бути результат одиниця, але якщо такий результат є, то всі інші повинні мати за результат нуль, оскільки одиниця для конкретного ресурсу вказує на те, що всі запити на використання в операційній системі за час її роботи від моменту запуску, були надіслані до ядра лише на виконання тільки цього ресурсу. Отримані дані також зберігаються в створеній структурі для проміжних даних.

Найголовніша та фінальна для отримання даних процедура є функція композиції відношень. Для цієї функції на вхід поступають дані від проміжної структури отриманих даних, яка вказувалась вище, згідно композицію відношень, яка вказана на рис. 3.1, за допомогою циклів перебираються всі дані процесного часу, процесного запиту до ресурсів та кількості запитів до кожного ресурсу. Початковими даними для композиції є дані процесного часу, спочатку за допомого цикла, кожен параметр масива перебирається з кожним параметром масива ресурсних запитів, та знаходить співвідношення між найбільш активним процесом та найбільш запитуваним ресурсом, потім від такого зв’язку виконується сама композиція та перевіряє за допомогою циклу їх зв'язок з любим іншим активним процесом, користуючись даними

процесного запиту. Таким чином через кожен найбільш запитуваний ресурс встановлюється зв’язок між найбільш активним процесами, коли від таких процесів відбувається чергове звернення до цього ж ресурсу, то програма половина подавати сигнал планувальнику про загрозу виникнення взаємного блокування, та вказує між якими процесами, та через який ресурс, і після цього планувальник повинен прийняти рішення на користь того, чи іншого процесу, підвищуючи, чи знижуючи пріоритет конкретного процеса. Тобто, композиція відношень шукає зв’язки між найбільш активними процесами через ресурс до його було надіслано багато запитів на користування, відносно всієї кількості запитів на використання, та встановлює зв’язок між ними. Потім визначені процеси та ресурс передаються сигналом планувальнику.

Головна функція описує порядок роботи інших функцій. Спочатку вона визначає який саме був системний виклик і порівнює його дані, з даними про

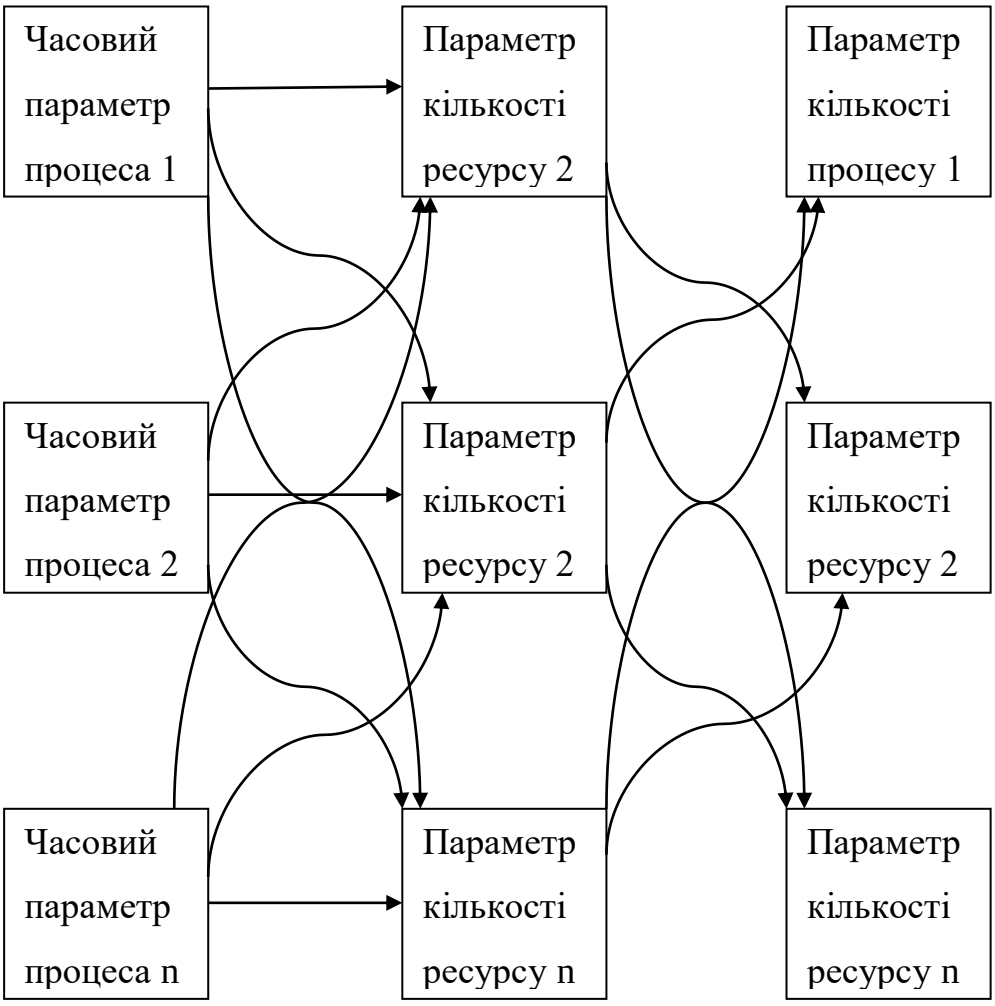

Рисунок 3.2 Схема композиції множин процесів та ресурсів

системні виклики які цікавлять дану програму, які зберігаються в двомірному масиві, той системний виклик. Через інструкцію switch(), знаходяться спів падання між системний викликом який відбувся та всіма системними викликами, які зберігаються у вказаному двомірному масиві, якщо спів падання є, то викликається певний набір функцій в залежності від системного виклику для занесення даних у структури процесів та ресурсів. Після цього викликаються функції обробки даних структур процесів та ресурсів та зберігаються у тимчасовій структурі. В останню чергу викликається функція композиції відносин, яка і відає кінцевий результат прогнозу ситуації взаємних блокувань між сказаними процесами та ресурсом.

3.4. Порівняльна характеристика методів

В даному розділі буде розглянуто найпоширеніші методи вирішення задачі взаємних блокувань з метою порівняння з розробленим методом. При порівнянні потрібно прийняти до уваги дуже важливу відмінність розробленого методу з аналогами вирішення задачі, яка заключається в тому, що вказаний метод єдиний, що проводить прогноз взаємного блокування на відміну від вже існуючих. Порівнюватись метод прогнозування буде з такими методами: метод організації черги, метод початкового запиту всіх ресурсів, метод відбору ресурсів, метод порядкової нумерації ресурсів.

Метод організації черги передбачає введення єдиного процеса, який фактично отримує можливість подавати запити на використання ресурсу. Всі інші процеси, організовуються в чергу на запит використання ресурсу. Недоліком розробленого алгоритму є те, що вводяться додаткові процеси, що працюють в фоновому режимі, та додаткові алгоритми обчислення, що ускладнюють роботу всієї операційної системи. До того ж, якщо немає запиту

на конкретний ресурс, то, спеціально створений для обробки запитів процес, просто буде знаходитися в стані блокування. Також, даний алгоритм породжує проблему виникнення взаємного блокування роботою планувальника операційної системи, який може в любий момент перервати роботу процесу, який користується ресурсом, та передати пріоритет на робочий час іншому процесу. Такі дії можуть також призвести до ситуації взаємного блокування.

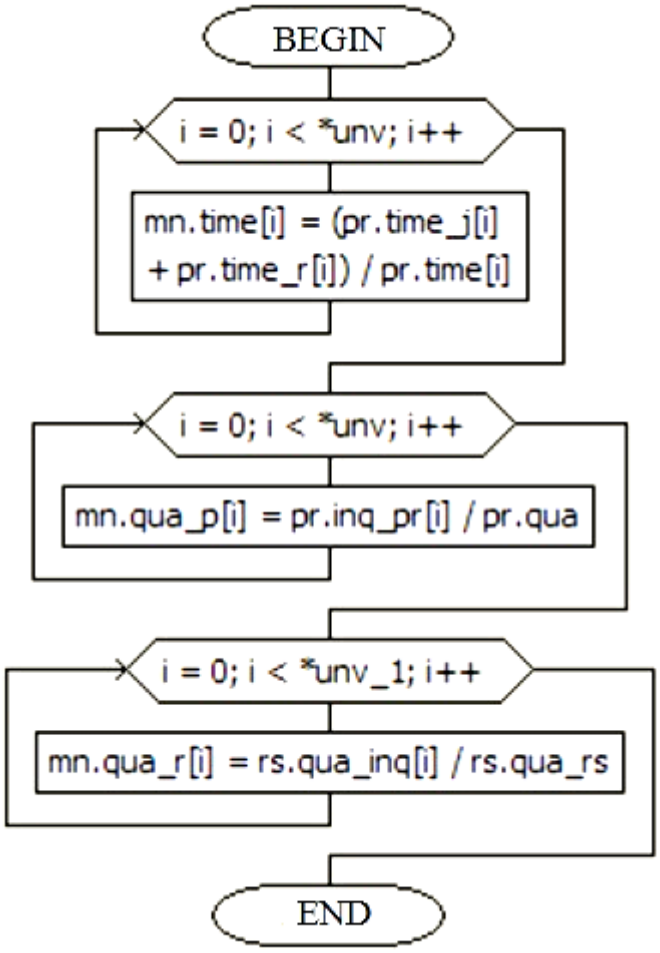

Рисунок 3.3 Алгоритм обробки створення нового процесу

Метод відбору ресурсів передбачає що у процеса, який зайняв ресурс заради свого виконання, неможливо відібрати даний ресурс, поки він не завершив з ним роботу. Це також виключає ситуацію взаємного блокування, але також має ряд серйозних недоліків. Щоб записи певних таблиць та баз даних всередині операційної системи можна було використовувати, то

потрібно їх заблокувати, тому потенційно може виникнути вірогідність взаємного блокування. Ще одним недоліком вказаного методу є те, що процеси вимушені будуть очікувати, поки ресурси, на використання яких було ними подано запит, будуть звільнені іншими процесами. Це розтягне чергу до найбільш запитуваних ресурсів та , відповідно, знизить продуктивність роботи більшості працюючих процесів в операційній системі. Тому цей метод також є неефективним на практиці.

Метод порядкової нумерації ресурсів передбачає два способи його реалізації. Перший спосіб передбачає, що процес може займати лише один ресурс в конкретний момент часу. Зрозуміло, що даний спосіб значно знижує продуктивність абсолютно всіх процесів в операційній системі, тому він є неприйнятним. Другий спосіб передбачає повної нумерації всіх запитів на використання ресурсів та впорядкований порядок звернення від всіх процесів до всіх ресурсів за зростання чисельного значення номеру запита до ресурса. При найбільш вищому номері запитуваного ресурса, який дав йому інший процес, неможливе надання процесові запитуваного ресурса, поки процес, який займає даний ресурс не звільнить його. Пронумерований порядок запитів від кожного процеса до ресурсів, дає можливість організації виключення взаємних блокувань при менших витратах ефективності роботи операційної системи, але також має ряд недоліків. Неможливо підібрати порядок нумерації, який задовольняє абсолютно всі процеси, тому що для всіх процесів, що залежні від ресурсів, є ряд найбільш важливих ресурсів, які включають в себе, наприклад, дисковий простір, є найбільш запитуваними ресурсами серед процесів, і без їх регулярного використання продуктивна для більшості процесів просто неможлива, що в свою чергу, значно знизить продуктивність роботи операційної системи.

При порівнянні всіх вище вказаних методів потрібно врахувати такі критерії:

- Зменшення ризику виникнення ситуації взаємного блокування
- Зменшення ефективності роботи процесів

- Додаткове навантаження на роботу планувальника
- Створення методом умови виникнення взаємних блокувань іншого типу.

Кожен критерій буде оцінюватись експертно за десяти бальною шкалою,чим вище бал, тим гірша ефективність методу. Перший, вказаний критерій, передбачає оцінки безпосереднього виконання своїх функцій кожним методом. Метод організації черги, впорядковуючи запити на ресурс всіма процесами, не може запобігти конкуренції процесів за ресурси, тому вона проявляється у формуванні черги, що і призводить до виникнення взаємних блокувань іншого типу. Метод початкового запиту всіх ресурсів повність виключає вірогідність виникнення взаємних блокувань. Метод відбору ресурсів, вводячи правило того, що неможливо в процеса відібрати ресурс, яким він користується, також повністю виключає можливість виникнення взаємного блокування. Метод порядкової нумерації ресурсів, впорядкувавши всі запити на використання процесів унеможливлює виникнення ситуації взаємного блокування. Метод прогнозування взаємних блокувань обраховуючи запити від найбільш пріоритетних процесів до найчастіше запитуваного ресурса ще в той момент часу, коли формується черга, також виключає умову виникнення ситуації взаємного блокування.

Другий критерій вказує на негативний вплив розроблених методів на роботу процесів в операційній системі. Метод організації черги та метод початкового запиту ресурсів значно підвищують час перебування процесів у стані очікування, а тому значно зменшують ефективність їх роботи. Методи відбору ресурсів менш негативно впливає на роботу діючих процесів, а методи порядкової нумерації ресурсів та прогнозування взаємних блокувань взагалі не втручаються в роботу процесів, тому ніякого негативного впливу на ефективність роботи процесів від вказаних методів немає.

Критерій додаткового навантаження на роботу планувальника вказую, що всі методи, з якими порівнюється метод прогнозування взаємних блокувань,

мають значний вплив на роботу планувальника, особливо метод порядкової нумерації ресурсів. Методи початкового запиту всіх ресурсів та відбору.

Таблиця 3.1

Порівняльна характеристика методів

	Зменшення ризику виникнення ситуації взаємного блокування	Зменшення ефективності роботи процесів	Додаткове навантаження на роботу планувальника	Створення методом умови виникнення взаємних блокувань іншого типу
Метод організації черги	3	7	3	3
Метод початкового запиту всіх ресурсів	0	7	6	0
Метод відбору ресурсів	0	3	5	2
Метод порядкової нумерації ресурсів	0	0	8	0
Метод прогнозування ситуації взаємних	0	0	1	0

блокувань				

ресурсів мають менш значний вплив на роботу планувальника, а метод організації черги незначний. Метод прогнозування взаємних блокувань втручається в роботу планувальника лише тоді, коли виникає ризик виникнення ситуації взаємного блокування та подає йому сигнал, в якому конкретно вказує ресурс та процеси, які можуть потрапити у вказану ситуацію, тому його вплив на роботу планувальника менш значний.

За четвертим критерієм, всі існуючі методи, запобігаючи ситуації взаємних блокувань не можуть запобігти конкуренції процесів, тому вони, усуваючи одні умови виникнення ситуації взаємних блокувань, утворюють інші умови виникнення ситуації взаємних блокувань. Тільки методи відбору ресурсів та організації черги створюють такі умови, всі інші методи не породжують додаткових умов для виникнення ситуації взаємних блокувань.

Зі всього вищевказаного можна зробити висновок, що саме метод прогнозування ситуації взаємних блокувань найкраще виконує функції вирішення вказаної задачі при цьому, не впливаючи на ефективність роботи процесів, та не створює додаткової роботи для планувальника.

3.5. Висновки

На базі запропонованого методу прогнозування ситуацій взаємних блокувань було розроблено програмний засіб, який оціню вірогідність виникнення взаємних блокувань. Програма була реалізована мовою програмування С, оскільки для повного контролю роботи планувальника та роботи процесів операційної системи потрібно щоб, програма чітко взаємодіяла з ядром, тобто була невід’ємною його частиною, а практично всі сучасні ядра операційних систем реалізовані мовою С та асемблер.

В програмному засобі було створено структури даних для процесів та ресурсів, які зберігають кількості запитів до кожного ресурса на використання, час роботи кожного процеса в певному стані та інші важливі дані. Також було розроблено функції обробки системних викликів, які відповідають за звернення від процеса до ресурса, та функціє оцінки кількості звернень від кожного процеса за час його існування, та до кожного ресурсу, та оцінки ризику виникнення взаємного блокування під час звернення процеса до конкретного ресурса.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1. Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Хошаба Олександр Мирославович та Богач Ілона Віталіївна

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою.

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Експерт 1	2. Експерт 2
	Бали, виставлені експертами:	
1	4	4
2	3	3
3	4	4
4	4	4
5	3	3
6	3	4
7	4	3
8	3	4
9	4	4
10	4	4

11	3	4
12	3	4
Сума балів	$СБ_1 = 43$	$СБ_2 = 45$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 44$	

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу.

4.2. Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи.

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

- кількість робочих днів у місяці, = 22 дні;

t - число днів роботи розробника, t = 45 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2 – Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	6000	272,72	5	1363,6
Інженер-програміст	3500	159,09	45	7159,05
Всього:				8522,65

Розрахуємо додаткову заробітну плату:

$$З_{дод}= 0,1\cdot 8522,65=852,26 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати:

$$Н_{зп}= (З_о + З_р)\cdot \frac{\beta}{100}, \quad (5.2)$$

$$Н_{зп}= (8522,65+852,26)\cdot \frac{36,3}{100} = 3403,09 \text{ (грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А= \quad (5.3)$$

де Ц – балансова вартість обладнання, грн;

– річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.3 – Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань,грн
Персональний комп’ютер	10000	25	3	625
Всього:				625

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n H_i \cdot] \quad (5.4)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Пачка паперу	уп.	150	1	150
Ручка	шт.	10	1	10
Всього з урахуванням транспортних витрат				176

Витрати на силову електроенергію розраховуються за формулою:

$$B_e = E ; \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,7 грн/кВт);

П – установлена потужність комп’ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп’ютера (Ф=150 год.);

– коефіцієнт використання потужності (< 1, $К_п$).

$$В_е = 1,7 \cdot 0,6 \cdot 150 \cdot 0,9 = 137,7 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (1..3) \cdot (З_о + З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1 * (8522,65 + 852,26) = 9374,91 \text{ (грн).}$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

В = 8522,65+852,25+3403,09+625+176+137,7+9374,91= 23091,61 (грн.)

Розрахуємо загальну вартість наукової роботи $В_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{23091,61}{1} = 23091,61$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$3I \qquad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{23091,61}{0,9} = 25657,34 \text{ (грн.)}$$

4.3. Прогнозування комерційних ефектів від реалізації результатів розробки.

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + \qquad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технологіїзменшатьсяна 25 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 25 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 200 користувачів, протягом другого року – на 175 користувачів, протягом третього року – 150 користувачів. Реалізація інформаційної технологіїдо впровадження результатів наукової розробки складала 1000 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 200 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 25 \cdot 1000 + (200 + 25) \cdot 200 = 70000 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 25 \cdot 1000 + (200 + 25) \cdot (200 + 175) = 109375 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 25 \cdot 1000 + (200 + 25) \cdot (200 + 175 + 150) = 143125 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність вкладених інвестицій розраховується за формулою:

$$E_{абс} = (ПП \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

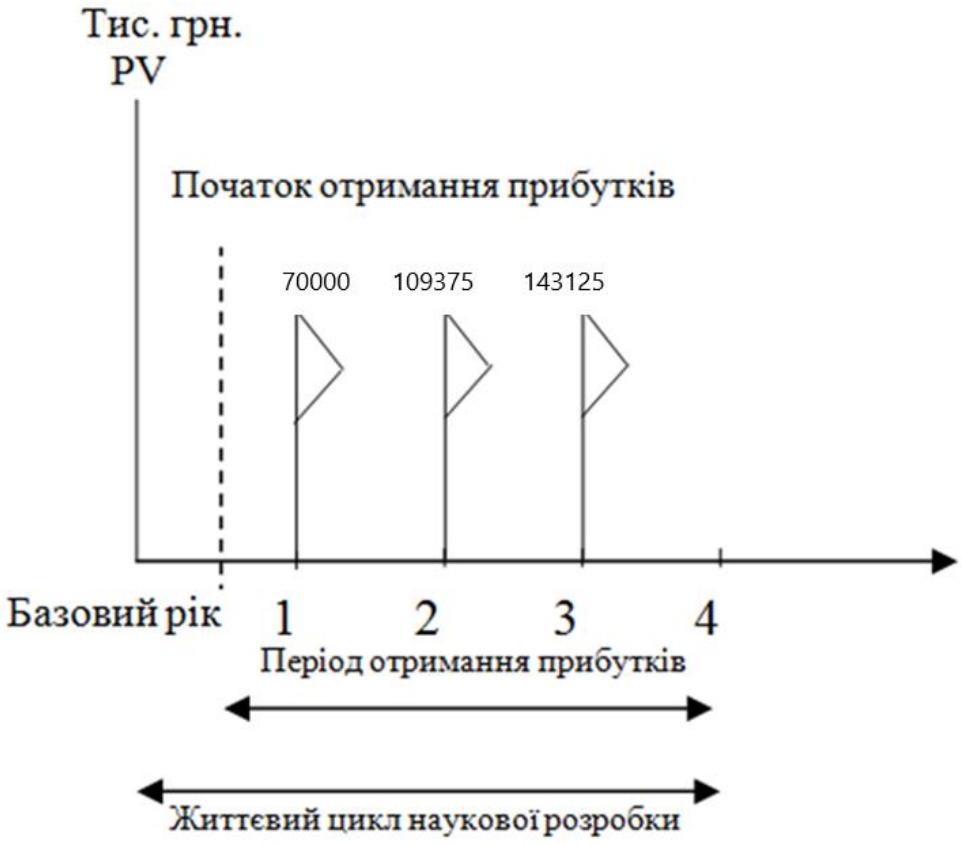

Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{25657,34}{(1+0,1)^0} + \frac{70000}{(1+0,1)^2} + \frac{109375}{(1+0,1)^3} + \frac{143125}{(1+0,1)^4} = 263 \text{ (грн.)}$$

Тоді розрахуємо

$$Е_{абс} = 263439,92 - 25657,34 = 237 \text{ грн.}$$

Оскільки E_{ab}, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$ за формулою:

$$E_в = \sqrt[T]{1 + } \quad (5.12)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$T_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$E_в = \sqrt[3]{1 + \frac{237782,58}{25657,34}} - 1 \text{ або } 117\ \%$$

Далі, розраховану величина $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau =,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0,2 + 0,1 = 0,3$$

Оскільки $E_в$ = 117% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $Т_{ок}$розраховується за формулою:

$$Т_{ок} = \frac{1}{Е_в}$$

$$Т_{ок} = \frac{1}{1,17} = 0,85 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

4.4. Висновки

Залучено незалежних експертів для проведення комерційного потенціалу розробки. За результатами оцінки проект показав комерційний потенціал вище середнього. Проведено порівняльний аналіз технічних показників розроблюваного проекту із іншими проектами для вирішення поставленої задачі. За результатами аналізу визнано, що нова розробка перевершує існуючі аналоги.

Виконано прогнозування витрат на виконання на наукової роботи та впровадження її результатів. Загальні витрати на виконання та впровадження результатів наукової розробки дорівнюють 25657,34 грн. Розраховано ефективність вкладених інвестицій та періоду їх окупності. Абсолютна ефективність вкладених інвестицій рівна 237782,58 гривень; термін окупності – 0,85 року, що підтверджує доцільність розробки та впровадження на ринок програмного продукту.

ВИСНОВКИ

В результаті виконання магістерської кваліфікаційної роботи було здобуту наступні наукові та практичні результати:

1.Проаналізовано ситуації виникнення взаємних блокувань та методи вирішення даної задачі.

2.Розроблено метод прогнозування взаємних блокувань, який є аналогом методам запобігання взаємним блокуванням.

3.Запропонований метод не впливає на ефективність роботи процесів в операційній системі, на відміну від методів запобігання взаємним блокуванням, які, в свою чергу, значно збільшують обсяг роботи для кожного процесу, вводячи для процесів виконання програм додаткові інструкції.

4.Ситауції виникнення взаємних блокувань та вірогідність її виникнення оцінюється за допомогою чіткого визначення ресурсної залежності та процесорної залежності від всіх працюючих процесів в операційній системі.

5.Введено розгляд всіх ресурсів, що були задіяні всіма існуючими в конкретний момент часу процесами в операційній системі, оцінка кількості звернення до кожного ресурса, та потреба в використані кожного процеса від кожного ресурса в подальшому, після кожного звернення за допомогою системного виклику.

6.Усуненено негативний вплив на роботу планувальника в операційній системі. На прикладі циклічного алгоритму планування роботи процесів в операційній системі,було показано, що розроблений програмний засіб на базі запропонованого методу впливає на роботу планувальника лише в тому, випадку, коли виникає реальний ризик виникнення ситуації взаємного блокування, на відміну від аналогів, що завжди впливають на формування планувальником черги процесів при будь-якому плануванні після кожного переривання.

7.Метод прогнозування взаємних блокувань більш ефективний від аналогів. Оцінюючи роботу всіх процесів та задіяних ними ресурсів він, за допомогою частоті запитів між процесами з ресурсною залежністю, процесами з процесорною залежністю та найбільш часто запитуваними ресурсами, вказує ядру операційної системи на ризик виникнення взаємного блокування лише в тих ситуаціях коли це можливо.

Отже, поставлені задачі магістерської кваліфікаційної роботи були виконані в повному обсязі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1.Таненбаум Е. Сучасні операційні системи 4 видання. – СПб.: Пітер, 2015, - 1120с.

2. Шеховцов В.А. Операційні системи. – К.: Видавнича група ВНУ, 2005. – 576 с.

3. Ричард Петерсен. LINUX: руководство по операционной системе. Второе издание, переработанное и дополненное в двух томах, перевод. – Киев: BHV, 1998, 1000 с.

4. Charles M. Shub A unified treatment of deadlock // Journal of Computing Sciences in Colleges Volume 2003 194-204 p.

5. Єфименко В.В., Оніщенко С.М., Франчук В.М. Операційні системи. Лабораторний практикум: Навчальний посібник. – К.: НПУ імені М.П. Драгоманова, 2008. – 124 с.

6. Martin Steffena Deadlock checking by a behavioral effect system for lock handling // The Journal of Logic and Algebraic Programming 2012 331–354 p.

7. Головина О.С., Кондратьев В.К. Операционные системы и оболочки. М: МЭСИ. 2002. – 434 с.

8. Андрей Лобачевский. Операционная система UNIX. BHV-СанктПетербург. 1997, 500 с.

9. F. Montesi, D. Sangiorgi, A model of evolvable components, in: Proceedings of the 5th International Symposium on Trustworthly Global Computing,

TGC 2010, in: Lecture Notes in Computer Science, vol. 6084, Springer, 2010, pp. 153–171.

10. Илюшкин Б.И. "Операционные Системы. Процессы и потоки" — Санкт-Петербург: 2005.

11. Э. Таненбаум, А. Вудхалл. «Операционные системы: Разработка и реализация.» — СПб.: 2006. – 1027 с.

12. Deitel H., Deitel P. and D. Choffnes. Operating Systems. Prentice Hall, 2003.

13. Столлингс У. Операционные системы = Operating Systems: Internals and Design Principles. — М.: Вильямс, 2004. — 848 с.

14. Рэймонд Э. С. Искусство программирования для UNIX = The Art of UNIX Programming. — М.: Вильямс, 2005. — 544 с.

15. Mark G. Sobell. UNIX System V. A Practical Guide. — 3rd ed. — 1995

16. Роберт Лав. Разработка ядра Linux = Linux Kernel Development. — 2-е изд. — М.: «Вильямс», 2006. — С. 448.

17. Ricard Marxer The impact of the Lombard effect on audio and visual speech recognition Systems // The Journal of Logic and Algebraic Programming 2013 34 p.

18. M. Bernardo, P. Ciancarini, L. Donatiello, Architecting families of software systems with process algebras, ACM Transactions on Software Engineering and Methodology 2002 386–426 p.

19. Ослэндер Д. М., Риджли Дж. Р., Рингенберг Дж. Д. Управляющие программы для механических систем: Объектно-ориентированное проектирование систем реального времени. — М.: Бином. Лаборатория знаний, 2004. — 416 с.

20. Квиттнер П. Задачи, программы, вычисления, результаты. — М.: Мир, 1980. — 422 с

ДОДАТКИ

Додаток А. Технічне завдання

Технічне завдання на магістерську кваліфікаційну роботу
Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О.Н.Романюк
"____ "_____2019р.

Технічне завдання
на магістерську кваліфікаційну роботу за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доц. О.М.Хошаба
"____"_____2019 р.
Виконав:
_____студент гр.1ПІ-18м К.С. Завертайло
"____"_____2019 р.

Вінниця – 2019 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методу і програмних засобів для прогнозування ситуації взаємних блокувань».

Галузь застосування – операційній системи.

2. Підстава для розробки.

Підставою для розробки магістерської кваліфікаційної роботи є індивідуальне завдання на МКР та наказ № ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є розробка методу прогнозування для зниження ризику виникнення ситуації взаємних блокувань, зменшення навантаження на роботу планувальника і підвищення ефективності роботи процесів в операційній системі.

Призначення роботи – розробка методів і засобів прогнозування ситуації взаємних блокувань.

4. Джерела розробки:

1.Таненбаум Е. Сучасні операційні системи 4 видання. – СПб.: Пітер, 2015, - 1120с.

2. Шеховцов В.А. Операційні системи. – К.: Видавнича група ВНУ, 2005. – 576 с.

3. Ричард Петерсен. LINUX: руководство по операционной системе. Второе издание, переработанное и дополненное в двух томах, перевод. – Киев: BHV, 1998, 1000 с.

4. Charles M. Shub A unified treatment of deadlock // Journal of Computing Sciences in Colleges Volume 2003 194-204 p.

5. Технічні вимоги

Методи розробки планувальників в операційних системах, методи розробки міжпроцесної взаємодії в операційних системах, методи розробки системних викликів в операційних системах, методи розробки ядра операційної системи, теорія множин та нечітких множин, методи приведення до фазифікації та дефазифікації нечітких множин.

6. Конструктивні вимоги.

Програмні засоби ядра UNIX-подібної операційної системи. Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз сучасних методів виявлення та запобігання взаємним блокуванням	07.09.2019 -27.09.2019
2	Розробка методу прогнозування взаємного блокування	28.09.2019 – 28.10.2019
	Розробка програмного забезпечення	29.10.2019 – 15.11.2019

3	для методу прогнозування взаємних блокувань	
4	Економічна частина	16.11.2019 – 29.11.2019

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б. Лістинг програмного засобу прогнозування ситуації взаємних блокувань

```
#include<stdio.h>
#include<string.h>

int *unv = 0;
int *unv_r = 0;
char name_prc[20];
char name_rs[20];
int t[2];

struct Pr
{
	char name_pr[100][20];
	int status_pr[100];
	int time[100];
	int time_r[100];
	int time_j[100];
	int n_p;
}pr;
```

```
struct Res
{
	char name_res[100][20];
	int kl_ob[100];
	int kl_ob_pr_to_r[100];
	int kl_obr;
	int n_p;
	int prc_to_res[100][100];
}rs;

struct MN
{
	float time[100];
	float pr_t_r[100];
	float to_r[100];
	float r_to_pr[100];
} mn;

struct mn_sort
{
	float time[100];
	float pr_t_r[100];
	float to_r[100];
	float r_to_pr[100];
	char res_sort[100][20];
	char prc_sort[100][20];
}mnr;

struct OTN
{
```

```
        float one[100];
        float two[100];
        float three[100];
        float four[100];
}otn;

void f_fork()
{
        unv = &pr.n_p;

        *unv++;

        for(int i = 0; i < 20; i++)
        {
                pr.name_pr[*unv][i] = name_prc[i];
        }

        rs.prc_to_res[*unv_r][*unv] = 0;

}

void f_kill()
{
        int a;

        for(int i = 0; i < pr.n_p; i++)
        {
                if(strcmp(pr.name_pr[i], name_prc)== 0)
                {
                        a = i;
```

```
                }
        }
        for(int j = a; j < pr.n_p; j++)
        {
        for(int i = 0; i < 20; i++)
        {
                pr.name_pr[j][i] = pr.name_pr[a++][i];
        }
           pr.time[j] = pr.time[a++];
           pr.time_j[j] = pr.time_j[a++];
           pr.time_r[j] = pr.time_r[a++];
    }

    for(int i = 0; i < 100; i++)
    {
        for(int j = 0; j < 100; j++)
        {
                rs.prc_to_res[i][j] = rs.prc_to_res[i++][j++];
        }
    }
}

void f_st_pr( int a)
{
        for(int i = 0; i < 100; i++)
        {
                if(strcmp(pr.name_pr[i], name_prc) == 0);
                {
                        pr.status_pr[i] = a;
                }
```

```
            if (pr.status_pr[i] != 1 || pr.status_pr[i] != 2)
            {
                pr.status_pr[i] == 0;
            };
        }
}

void f_tim()
{
        int a;
        a = t[1] - t[0];

        for(int i = 0; i < 100; i++)
        {
            pr.time[i] = pr.time[i] + a;
            pr.time_j[i] = pr.time_j[i] + a;
            pr.time_r[i] = pr.time_r[i] + a;
        }
}

void f_rwo()
{
        int x = 0;

        for(int i = 0; i < rs.n_p; i++)
        {
            if(strcmp(rs.name_res[i], name_rs) == 0)
            {
                rs.kl_ob[i]++;
                rs.kl_obr++;
```

```
                x = i;
            } else
            {

                for(int j = 0; j < 20; j++)
                {
                    rs.name_res[rs.n_p][j] = name_rs[j];
                        rs.n_p++;
                        rs.kl_obr++;
                }

            }
        }

        for(int i = 0; i < *unv; i++)
        {
            if(strcmp(pr.name_pr[i], name_prc))
            {
                if(rs.prc_to_res[x][i] == 0)
                {
                    rs.kl_ob_pr_to_r[x]++;
                }
                rs.prc_to_res[x][i]++;
            }
        }


}

void f_ras()
```

```
{
	for(int i = 0; i < *unv; i++)
	{
		mn.time[i] = (pr.time_r[i] + pr.time_j[i])/pr.time[i];
	}

	for(int i = 0; i < *unv_r; i++)
	{
		for(int j = 0; j < *unv; j++)
		{
		mn.pr_t_r[i] = rs.prc_to_res[i][j]/rs.kl_ob[i];
	}
	}

	for(int i = 0; i < *unv_r; i++)
	{
		mn.to_r[i] = rs.kl_ob[i]/rs.kl_obr;
	}

	for(int i = 0; i < *unv_r; i++)
	{
		mn.r_to_pr[i] = rs.kl_ob_pr_to_r[i]/pr.n_p;
	}
}

void f_ras_2()
{
	for(int i = 0; i < 100; i++)
	{
		otn.one[i] = mn.time[i]/mn.to_r[i];
```

```
        }

        for(int i = 0; i < 100; i++)
        {
                otn.two[i] = mn.time[i]/mn.r_to_pr[i];
        }

        for(int i = 0; i < 100; i++)
        {
                otn.three[i] = mn.pr_t_r[i]/mn.to_r[i];
        }

        for(int i = 0; i < 100; i++)
        {
                otn.four[i] = mn.pr_t_r[i]/mn.r_to_pr[i];
        }
}

void d_ras_3()
{
        for(int i = 0; i < *unv; i++)
        {
                for(int j = 0; j < 20; j++)
                {
                        mnr.prc_sort[i][j] = pr.name_pr[i][j];
                        mnr.res_sort[i][j] = rs.name_res[i][j];
                }
        }

        for(int i = 0; i < *unv; i++)
```

```
{
        mnr.pr_t_r[i] = mn.pr_t_r[i];
        mnr.r_to_pr[i] = mn.r_to_pr[i];
        mnr.time[i] = mn.time[i];
        mnr.to_r[i] = mn.to_r[i];
}
for(int i = 0; i < *unv; i++)
{
        for(int j = 0; j < *unv; j++)
        {
                if(mnr.pr_t_r[j] > mnr.pr_t_r[i])
                {
                        int a;
                        a = mnr.pr_t_r[i];
                        mnr.pr_t_r[i] = mnr.pr_t_r[j];
                        mnr.pr_t_r[j] = a;
                }
                if(mnr.r_to_pr[j] > mnr.r_to_pr[i])
                {
                        int a;
                        a = mnr.r_to_pr[i];
                        mnr.r_to_pr[i] = mnr.r_to_pr[j];
                        mnr.r_to_pr[j] = a;
                }
                if(mnr.time[j] > mnr.time[i])
                {
                        int a;
                        a = mnr.time[i];
                        mnr.time[i] = mnr.time[j];
                        mnr.time[j] = a;
```

```
                }
                if(mnr.to_r[j] > mnr.to_r[i])
                {
                        int a;
                        a = mnr.to_r[i];
                        mnr.to_r[i] = mnr.to_r[j];
                        mnr.to_r[j] = a;
                }
        }
}

int a = 2;

if(*unv > 10)
{
        a = 5;
}else if (*unv > 30)
{
        a = 10;
}else if (*unv > 50)
{
        a = 15;
}

int b = 2;

if(*unv > 10)
{
        b = 5;
}else if (*unv > 30)
```

```
        {
                b = 10;
        }else if (*unv > 50)
        {
                b = 15;
        }

        for(int i = 0; i < a; i++)
        {
                for(int j = 0; j < b; j++)
                {
                        for(int z = 0; z < b; z++)
                        {
                                if( z == j) continue;
                                else if(rs.prc_to_res[z][i] > 0 || rs.prc_to_res[j][i] > 0)
                                {
                                        printf("Deadlock");
                                }
                        }
                }
        }
}


int main()
{

        char sys_c_k[5] = {'k', 'i', 'l', 'l', '\0'};
        char sys_c_f[5] = {'f', 'o', 'r', 'k', '\0'};
    char sys_c_o[5] = {'o', 'p', 'e', 'n', '\0'};
```

```
char sys_c_r[5] = {'r', 'e', 'a', 'd', '\0'};
char sys_c_w[6] = {'w', 'r', 'i', 't', 'e', '\0'};
    char sys_c_s[7] = {'s', 'e', 't', 's', 'i', 'd', '\0'};
    char sys_c_e[5] = {'e', 'x', 'i', 't', '\0'};

    char sys_c[10];

    int y;

    FILE *system;

    while(!feof(system))
    {
        fgets(sys_c, 20, system);
        fgets(name_prc, 20, system);
        fgets(name_rs, 20, system);
        t[1] = getc(system);
        y = getc(system);
    }

    f_tim();


    switch(sys_c[0])
    {
        case 'w': f_rwo(); break;
        case 'r': f_rwo(); break;
        case 'o': f_rwo(); break;
        case 'f': f_fork(); break;
        case 's': f_st_pr(y); break;
```

```
            case 'e': f_st_pr(y); break;
            case 'k': f_kill(); break;
        }
        f_ras();  f_ras_2();   return 0;  }
```

Додаток В. Ілюстративний матеріал

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д. т. н., професор _____ О. Н. Романюк

Науковий керівник, к. т. н., доцент кафедри ПЗ_____ О.М. Хошаба

Рецензент, к.т.н, доцент кафедри КН_____ І.В. Богач

Нормоконтроль, к. т. н., доцент кафедри ПЗ _____О.М. Хошаба

Виконавець, студент групи 1ПІ-18м _____ К.С.