Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

До захисту допущено
Завідувач кафедри
комп'ютерної інженерії
к.т.н., доц. О.М.Березький
_____
"___" _____ 20___ р.

ДИПЛОМНА РОБОТА
освітньо-кваліфікаційного рівня "Магістр"
зі спеціальності 8.05010201 "Комп'ютерні системи та мережі"
на тему:

МАТЕМАТИЧНІ МОДЕЛІ ОРГАНІЗАЦІЇ ОБЧИСЛЕНЬ В КЛАСТЕРНИХ СИСТЕМАХ НА ОСНОВІ ВИБОРУ ОПТИМАЛЬНИХ ЗАСОБІВ ВЗАЄМОДІЇ ПРОЦЕСІВ

Студент групи КСМзм - 51
Бендас Ю.В.

_____
підпис
Науковий керівник
д.е.н., професор Ріппа С.П.

_____
підпис
Консультант з нормоконтролю
Палій І.О.
_____
Прізвище, ініціали Підпис

Тернопіль – 2012

Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

"Затверджую"
Зав. кафедри
комп'ютерної інженерії
к.т.н., доц. О.М. Березький
_____
"____"_____20__ р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТА
Бендаса Юрія Васильовича

1. **Тема дипломної роботи** "Математичні моделі організації обчислень в кластерних системах на основі вибору оптимальних засобів взаємодії процесів" затверджена наказом університету №_____ від „____" _____20__ р

2.**Термін здачі** закінченої дипломної роботи_____

3. **Об'єкт дослідження**: обчислювальні процеси в комп'ютерних системах з багатоядерною архітектурою.

4. **Предмет дослідження**: математичні моделі обчислень в кластерних системах з багатоядерною архітектурою.

5. **Перелік задач, які мають бути вирішені**:
- аналіз апаратного забезпечення кластерних систем та визначення основних чинників впливу на організацію обчислень в КСБА;
- аналіз особливостей організації обчислювальних процесів в КСБА, пов'язаних з ускладненням структурної організації системи;
- розробка моделей обчислень, що дозволяють описати поведінку і взаємодію процесів в КСБА на рівні вузлів системи при різній організації системи зв'язків вузлів;
- розробка моделей обчислень, що дозволяють описати поведінку і взаємодію процесів у вузлах КСБА, що враховують багатоядерну архітектуру вузла;
- реалізація моделей обчислень в кластерних системах з багатоядерною архітектурою;
- дослідження тупикових ситуації при використанні об'єктів комунікації і синхронізації в КСБА.

6. **Перелік ілюстративного матеріалу:**
- схеми комутації вузлів кластерної системи,
- схеми взаємодії вузлів КСБА,
- структура об'єкта «вузол»,
- схема зв'язування об'єктів в моделях КСБА1 і КСБА2,
- схема взаємодії в КСБА1 на основі RPC,
- схема взаємодії задач в механізмі рандеву,
- схеми синхронізації процесів,
- схеми тупикових ситуацій в механізмі рандеву,
- схеми безтупикових ситуацій в механізмі рандеву.

7.**Консультанти** по роботі

Розділ	Консультант	Підпис
1		
2		
3		

КАЛЕНДАРНИЙ ПЛАН

№	Назва структурних частин ДР	Термін виконання	Примітка
1	Огляд методів і засобів організації обчислень в кластерних системах	15.09.2011 – 5.11.2011	
2	Математичні моделі організації обчислень в кластерних системах з багатоядерною архітектурою	6.11.2011 – 31.01.2012	
3	Реалізація моделей обчислень	1.02.2012 – 23.04.2012	

Завдання прийняв до виконання _____

(підпис)

Керівник дипломної роботи _____

(підпис)

РЕФЕРАТ

Дипломна робота на тему “Математичні моделі організації обчислень в кластерних системах на основі вибору оптимальних засобів взаємодії процесів” на здобуття освітньо-кваліфікаційного рівня “Магістр” зі спеціальності “Комп’ютерні системи та мережі” написана обсягом 90 сторінок і містить 22 ілюстрації, 2 таблиці, 2 додатки та 39 джерел за переліком посилань.

Метою роботи є вдосконалення організації обчислювальних процесів в кластерних системах з багатоядерною архітектурою, що дозволяє повною мірою реалізувати усі переваги структурної організації кластерних систем.

Методи досліджень. В дипломній роботі при вирішенні поставлених завдань використані методи алгебри процесів для опису поведінки процесів, теорії послідовних взаємодіючих процесів (CSP теорії) для опису і аналізу різних видів взаємодії процесів.

Розроблені моделі організації обчислень для опису поведінки процесів на обох рівнях кластерних систем з багатоядерною архітектурою, що дозволяють оптимізувати їх взаємодію з врахуванням наявності різних форм взаємодії, що мають місце в кластерних системах з багатоядерною архітектурою.

Запропоновані способи організації обчислювальних процесів дозволяють підвищити ефективність обчислень в кластерних системах з багатоядерною архітектурою з різною структурною організацією.

Ключові слова: КЛАСТЕРНА СИСТЕМА, МОДЕЛЬ ОРГАНІЗАЦІЇ ОБЧИСЛЕНЬ, ВЗАЄМОДІЯ ПРОЦЕСІВ, БАГАТОЯДЕРНА АРХІТЕКТУРА, СТРУКТУРНА ОРГАНІЗАЦІЯ.

ABSTRACT

Diploma work «Mathematical models of computation in cluster systems based on the optimal choice of the interaction processes» on acquiring of educationally-qualification «Master's» degree, speciality «Computer Systems and Networks» has general volume 90 pages and contains 22 illustrations, 2 tables, 2 additions and 39 sources of information.

The object is to improve the organization of computational processes in cluster systems with multicore architecture, which allows to realize all the benefits of the structural organization of cluster systems.

Methods of research. In the thesis work during tasks salvation are used methods of process algebra to describe the behavior of processes and the theory of co-operation sequential processes (CSP theory) to describe and analyze different types of interaction processes.

Models of computation to describe the behavior of processes at both levels of multi-cluster systems architecture are developed. That enables to optimize their interactions with regard to the presence of different forms of interaction that occur in systems with multi-cluster architecture.

Proposed methods of computational processes enable to improve efficiency in computing systems with multi-cluster architecture with different structural organization.

Keywords: CLUSTER SYSTEM, MODEL OF COMPUTATION ORGANIZATION, CO-OPERATION PROCESSES, MULTICORE ARCHITECTURE, STRUCTURAL ORGANIZATION.

ЗМІСТ

ВСТУП

Актуальність роботи. Одним із способів оптимізації обчислювального процесу є його розпаралелювання з метою подальшої реалізації на системах паралельної архітектури. Для реалізації паралельних методів та алгоритмів використовуються обчислювальні засоби універсального та спеціального призначення. На даний час розвиток універсальних обчислювальних систем здійснюється за чотирма основними напрямками: векторноконвеєрні; SMP (Symmetric Multi-Processing); MPP (Massively Parallel Processing) та кластери [1].

Зараз спостерігається стійка тенденція стосовно поширення та застосування кластерних обчислювальних систем. Для побудови кластерів використовують компоненти, які серійно випускаються, забезпечують високу продуктивність обчислень та масштабованість обчислювальної системи. Очевидно, що зміни в елементній базі призводять до певних змін у підходах до побудови паралельних алгоритмів. Зауважимо, що кластери є дешевою альтернативою до дорогих суперкомп’ютерів. Недоліком останніх є те, що вони покращення ефективності обчислювальної системи завдяки збільшенню тактової частоти процесора практично вичерпані або є економічно не вигідними.

Тому перспективною є ідея підвищення продуктивності архітектури обчислювальних систем унаслідок збільшення кількості процесорів. Багатопроцесорні та багатоядерні обчислювальні системи все ширше використовуються у різних предметних областях і поступово витісняють однопроцесорні. Зараз триває процес модернізації кластерів шляхом переходу на багатоядерні процесори виробництва компаній AMD, Intel та IBM [2].

Організація обчислень в комп’ютерних системах з багатоядерною архітектурою (КСБА) пов'язана з вирішенням двох задач – організації ефективної взаємодії вузлів системи і обчислень в кожному вузлі, які у свою чергу вимагають ефективної реалізації обчислень в кожному ядрі і їх взаємодії.

У зв'язку з цим актуальною є розробка і вдосконалення моделей паралельних обчислень, що відображають особливості архітектури КСБА, які дозволяють скоротити час розробки програмного забезпечення для КСБА і

підвищити його якість.

Мета і завдання дослідження. Метою даної дипломної роботи є вдосконалення організації обчислювальних процесів в кластерних системах з багатоядерною архітектурою, що дозволяє повною мірою реалізувати усі переваги структурної організації КСБА.

Для досягнення поставленої мети необхідно виконати наступні основні завдання дослідження:

– аналіз апаратного забезпечення кластерних систем та визначення основних чинників впливу на організацію обчислень в КСБА;

– аналіз особливостей організації обчислювальних процесів в КСБА, пов'язаних з ускладненням структурної організації системи;

– дослідження моделей паралельних обчислень в КСБА;

– розробка моделей обчислень, що дозволяють описати поведінку і взаємодію процесів в КСБА на рівні вузлів системи при різній організації системи зв'язків вузлів;

– розробка моделей обчислень, що дозволяють описати поведінку і взаємодію процесів у вузлах КСБА, що враховують багатоядерну архітектуру вузла;

– реалізація моделей обчислень в кластерних системах з багатоядерною архітектурою;

– дослідження тупикових ситуації при використанні об'єктів комунікації і синхронізації в КСБА.

Об’єкт дослідження – обчислювальні процеси в комп'ютерних системах з багатоядерною архітектурою.

Предмет дослідження – математичні моделі обчислень в кластерних системах з багатоядерною архітектурою, а також організація обчислювальних процесів в таких системах.

Методи досліджень. В дипломній роботі при вирішенні поставлених завдань використані методи алгебри процесів для опису поведінки процесів, теорії послідовних взаємодіючих процесів (CSP теорії) для опису і аналізу різних видів взаємодії процесів.

Наукова новизна одержаних результатів. Вдосконалені моделі організації обчислень для опису поведінки процесів на обох рівнях КСБА, що дозволяють оптимізувати їх взаємодію з врахуванням наявності різних форм взаємодії, що мають місце в КСБА.

Практичне значення отриманих результатів. Запропоновані способи організації обчислювальних процесів дозволяють підвищити ефективність обчислень в КСБА з різною структурною організацією. Запропоновані математичні моделі і алгоритми доведені до практичної реалізації у вигляді програм і можуть бути використані при організації обчислювальних процесів в КСБА.

В першому розділі дипломної роботи виконаний аналіз особливостей організації обчислювальних процесів в КСБА. Показано, що система володіє двома рівнями паралельної обробки – між вузлами і на рівні кожного вузла і вимагає розробки моделей паралельних обчислень, що забезпечують опис поведінки процесів на обох рівнях.

В другому розділі розроблено математичні моделі обчислень першого та другого рівнів для комп’ютерних систем з багатоядерною архітектурою, які дозволяють описати дворівневу організацію обчислень в КСБА і виконати роль інтерфейсу між архітектурою системи та моделями програмування, що спрощує розробку програмних компонент для КСБА.

У третьому розділі розглянуті питання реалізації запропонованих моделей паралельних обчислень для КСБА. Реалізація виконується за допомогою спеціальних програмних засобів, які є в наявності в сучасних мовах програмування і бібліотеках програмування. В якості базової мови розглядається мова програмування Ада.

1 ОГЛЯД МЕТОДІВ І ЗАСОБІВ ОРГАНІЗАЦІЇ ОБЧИСЛЕНЬ В КЛАСТЕРНИХ СИСТЕМАХ

1.1 Апаратне забезпечення кластерних систем

Кластерна система (КС) - вид розподіленої обчислювальної системи, що є сукупністю комп'ютерів, об'єднаних за допомогою комп'ютерної мережі для вирішення однієї задачі. Представляється для користувача як єдиний обчислювальний ресурс. Концепція кластерної системи вперше була запропонована компанією DEC на початку 80-х років минулого століття [3]. На сьогодні це найбільш поширений тип комп'ютерних систем, що дозволяє масову реалізацію концепції паралельної обробки, оптимальну за критерієм ціна/продуктивність. Кластерна архітектура дозволяє реалізувати системи, які забезпечують обробку великих об'ємів даних (високопродуктивні кластерні системи), надійність, високу готовність, масштабованість, відмовостійкість.

Загальна структура кластерної системи включає набір вузлів і комунікаційне середовище. В якості обчислювального вузла використовуються звичайні комп'ютери, потужні робочі станції, 2-4-х процесорні СМП системи. Кожен вузол працює під управлінням своєї операційної системи (Linux, Windows, Solaris). Кількість і вид вузлів може змінюватися у рамках одного кластера залежно від задачі, що вирішується.

Традиційно класичною кластерною системою вважається система, створена у рамках проекту Beowulf, що стартував в 1994 році [4]. Вона включала 16 вузлів, кожен з яких був реалізований за допомогою процесора Intel 486DX4/100, оперативної пам'яті 16 Мб, трьох мережевих адаптерів. Призначення кластера - забезпечення обчислювальних потужностей для проекту Earth and Space Sciences в NASA.

Найважливішою складовою кластерної системи є комунікаційне середовище, що визначає ефективність системи при рішенні задачі обміну даними між вузлами системи. Основу комунікаційного середовище кластерної системи визначають мережеві технології, такі як Ethernet. Проте розвиток кластерних

систем привів до появи нових комунікаційних середовищ - Myrinet, Infiniband [5].

У даній роботі розглядаються підходи до організації обчислювальних процесів у високопродуктивних кластерних системах. Особливістю апаратної і програмної складових таких систем є забезпечення високошвидкісних обчислень, пов'язаних з великими об'ємами даних. Це можливо при використанні в системі потужних обчислювальних ресурсів, а також ресурсів, пов'язаних з організацією ефективної передачі даних. Забезпечення високої продуктивності кластерних систем також можливо тільки при ефективній організації обчислювальних процесів.

Структура сучасної КС включає дві основні складові - набір вузлів і систему зв'язків вузлів. Побудова ефективної КС припускає ефективну реалізацію обох складових, оскільки вони визначають в КС підтримку високопродуктивних обчислень.

Остання редакція рейтингу високопродуктивних комп'ютерних систем TOP500 включає 61 кластерну систему в перших ста позиціях [6]. Кількість процесорів в цих КС варіюється в діапазоні від 1992 до 122400, максимальна продуктивність - в діапазоні від 18,81 до 1026 Tflop/s, пікова продуктивність - в діапазоні від 24,58 до 1375,78 Tflops. На сьогодні TOP500 очолює кластерна система Roadrunner компанії IBM.

У таблиці додатку А представлені кластерні системи, що входять в TOP100. Системи відрізняються великою різноманітністю використовуваних апаратних засобів. Слід зазначити, що в порівнянні з 28-ою редакцією число КС зросло в 3 рази, а в порівнянні з 29-ою редакцією - в 4 рази. Це говорить про перспективність побудови високопродуктивних обчислювальних систем на основі кластерної архітектури.

Вузол кластерної системи - комп'ютер, робоча станція, багатопроцесорна система. Вузол повинен забезпечити основне завдання кластерної системи, пов'язане з виконанням високопродуктивних обчислень.

Типи використовуваних процесорів - звичайні Інтел і АМД процеси, потужні серверні процесори (Інтел), процесори, побудовані на RISC архітектурі.

Система зв'язків вузлів. Найважливішою складовою кластерної системи є

система зв'язків вузлів (СЗВ). Її завдання - забезпечити високошвидкісну передачу даних між вузлами, використовуючи ефективні комунікаційні технології

Продуктивність СЗВ характеризується двома параметрами: латентністю, що визначає час початкової затримки при передачі повідомлень, і пропускною здатністю, яка визначає швидкість передачі даних по каналах зв'язку.

На сьогодні в КС використовуються різні види технологій для реалізації системи зв'язків вузлів: Fast Ethertnet, Gigabit Ethernet, Myrinet, cLAN(Giganet), SCI, QsNetII (QSW), MEMORY CHANNEL, ServerNet II, InfiniBand, Flat Neighborhood:

- Fast Ethernet. Пікова пропускна здатність - 100 Mbit/sec (12.5 MB/sec), повний дуплекс. У рамках MPI досягаються швидкості близько 6-7 MB/sec. Перевагами цієї технології є хороша стандартизація і широке поширення, а також низькі ціни на устаткування і використання стандартних мережевих кабелів (UTP);

- Gigabit Ethernet. Пікова пропускна здатність - 1 Gbit/sec (125 MB/sec), повний дуплекс. У рамках TCP/IP досягаються швидкості близько 500 Mbit/sec (60 MB/sec), у рамках MPI - до 45 MB/sec. Перевагою цієї технології є сумісність і можливість плавного переходу з технологій Ethernet на Fast Ethernet;

- Myrinet 2000. Пікова пропускна здатність - 2 Gbit/sec, повний дуплекс. Апаратна латентність близько 5 мксек. У рамках TCP/IP досягаються швидкості близько 1.7-1.9 Gbit/sec (240 MB/sec). Латентність - близько 30 мксек. На MPI - додатках латентність складає близько 10 мксек, швидкість передачі даних - до 200 MB/sec (до 400 MB/sec на дуплексних операціях). Myrinet є відкритим стандартом. Myricom пропонує широкий вибір мережевого устаткування за порівняно невисокими цінами. На фізичному рівні підтримуються мережеві середовища SAN (System Area Network), LAN (CL - 2) і оптоволокно. Технологія Myrinet дає високі можливості масштабування мережі і нині широко використовується при побудові високопродуктивних кластерів;

- LAN. Пікова пропускна здатність - 1066 MB/sec. Продукти сімейства cLAN і MPI/Pro пропонуються у складі технології кластеризації Gigacluster;

- SCI (Scalable Coherent Interface). Для продуктів Dolphin: пікова пропускна

здатність - 10 GB/sec, у рамках MPI досягається близько 700 MB/sec. Апаратна латентність - 0.2 мксек, у рамках MPI - близько 1,4 мксек. SCI - стандартизована технологія (ANSI/IEEE l596 - 1992);

- QsNetII. Пікова пропускна здатність каналів - 1064 MB/sec (досягається 900 MBytes/sec в одному напрямі, і 1800 MB/sec в режимі Multi - rail). Латентність у рамках MPI - близько 1,5 мксек. Максимальний розмір системи - більше 4000 вузлів. Мережеве устаткування складається з комунікаційних процесорів "Elan" і матричних комутаторів 4x4 "Elite". На базі цих комутаторів будується єдине комунікаційне середовище з топологією "fat tree", підтримується глобальна адресація оперативної пам'яті. Комунікаційне устаткування QsNet використовується в системах AlphaServer SC від Compaq;

- MEMORY CHANNEL. Пікова пропускна здатність каналів більше 100 MB/sec, латентність - 3 мксек. Технологія MEMORY CHANNEL забезпечує функціональність віддаленого доступу до пам'яті інших вузлів в кластері (пам'ять інших вузлів може відображатися в локальний адресний простір). Використовується в кластерних системах AlphaServer Array і HPC320/HPC160 від Compaq;

- InfiniBand. Пропонує віддалений прямий доступ в пам'ять (remote direct memory access - RDMA), що дозволяє доставляти дані безпосередньо в пам'ять процесора, не залучаючи системні виклики. Дані можуть передаватися 1-о, 4-х і 12-ти кратною швидкістю. Латентність на комутаторі InfiniBand складає 160 наносекунд.

Аналіз кластерних систем, представлених в перших сто позиціях рейтингу TOP500 показав, що найбільш використовуваною комунікаційною технологією є InfiniBand (38 позицій). За нею йде технологія GigabitEthernet (10). У списку представлені також технології Myrinet (7) і QsNet II (2).

1.2 Організація обчислень в кластерних системах з багатоядерною архітектурою

Ріст продуктивності КС традиційно пов’язаний зі збільшенням кількості вузлів системи. Проте нарощування числа вузлів обмежене із-за підвищеної вимоги до системи зв'язків вузлів при збільшенні вузлів і навантаженні на систему зв'язків. Є другий шлях підвищення продуктивності КС, заснований на збільшенні обчислювальної потужності безпосередньо самого вузла. Такий підхід забезпечується використанням у вузлах потужних процесорів, також реалізацією у вузлах КС принципів паралельної обробки за рахунок використання багатопроцесорних систем. Використання у вузлах багатопроцесорних систем дозволяє перейти в КС до дворівневої паралельної обробки. Перший рівень забезпечується паралельною обробкою на множині вузлів КС, другий - на множині процесорів всередині вузла. Проте, паралелізм на рівні вузла на сьогодні використовується не повною мірою, оскільки вузли існуючих КС, як правило, побудовані на дорогих СМП системах, що включають 2 (рідше 4) процесори.

Поява багатоядерних процесорів відкриває перспективи збільшення потужності вузлів за рахунок ефективнішої реалізації другого рівня паралельної обробки в КС. Проста заміна у вузлі звичайного процесора на двохядерний процесор дозволяє подвоїти обчислювальну потужність КС без зміни її структури. Поява на ринку чотириядерних процесорів і анонсоване в найближчому майбутньому появи процесорів з вісьмома і більше ядрами, відкриває можливості різкого збільшення продуктивності КС при побудові на їх основі кластерних систем з багатоядерною архітектурою (КСБА).

Як видно з рейтингу TOP100, на сьогодні багатоядерні процесори використовуються вже в 62 кластерних системах. Серед них основну масу складають процесори Інтел. Також використовуються процесори від АМД і IBM.

Таким чином, розвиток кластерної архітектури пов'язаний з використанням багатоядерних процесорів. Це у свою чергу вимагає вдосконалення принципів організації обчислювальних процесів, завданням яких стане ефективне використання обох рівнів паралелізму вКСБА.

Метою даної роботи є вдосконалення організації обчислювальних процесів в кластерних системах з багатоядерною архітектурою, що дозволяє повною мірою реалізувати усі переваги структурної організації КСБА.

Збільшення кількості ядер ставить проблему організації зв'язків між ядрами. Шинна архітектура, яка використовувалася в 2-4 ядрах стане гальмом для ефективного зв'язку ядер і неминучим буде перехід на розподілену архітектуру. Так, наприклад, зв'язування ядер за допомогою топології типу грати дозволять відмовитися від централізованих ресурсів, а також розв'язати проблему надійності при виході з ладу ядер або зв'язків.

Випуск багатоядерних процесорів сьогодні виконується декількома компаніями. Лідерами є компанії Intel і AMD. Разом з ними випуском багатоядерних процесорів для систем серверного рівня займаються також компанії Sun і IBM. Їх особливість - реалізація RISC архітектури. Розглянемо особливості реалізації багатоядерних процесорів на прикладі архітектури чотириядерного процесора компанії Intel Core i7.

Процесор Core i7 (Bloomfield, 45 нм) включає (рисунок 1.1): чотири фізичні ядра, розподілених на 8 віртуальних потоків за допомогою технології Intel Hyper - Threading, трирівневий масив осередків кеш-пам'яті, новий системний інтерфейс Quick Path Interconnect, інтегрований контролер пам'яті DDR3, блок черги команд і блок операцій вводу-виводу. У структурі серверних чотириядерних процесорів Nehalem EP буде використаний другий порт QPI для зв'язку декількох процесорів одній материнській платі.

Дизайн Core i7 спочатку припускає наявність чотирьох ядер в одному процесорному кристалі. Кількість фізичних ядер легко варіюється залежно від наміченої мети. У другій половині 2009 р. компанія Intel випутила восьми ядерні CPU Nehalem EP.

Ядро Bloomfield має вбудований контролер оперативної пам'яті (IMC), кеш третього рівня, що розділяється, і високошвидкісний інтерфейс QPI. Перенесення контролера пам'яті з північного моста в тіло CPU дозволило зменшити залежність процесора від постійного збільшення об'єму кеш-пам'яті.

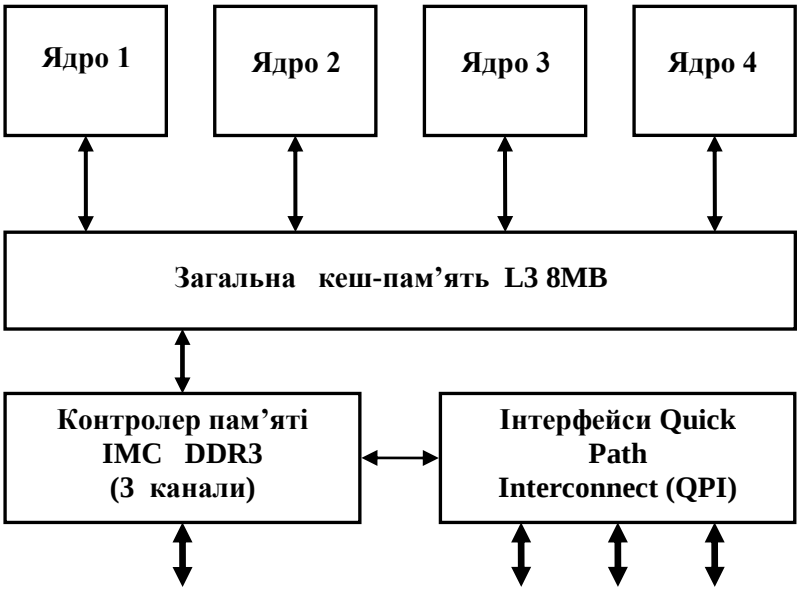


Рисунок 1.1 – Архитектура Nehalem процессоров Intel Core i7

Ієрархія кеш-пам'яті в процесорі Core i7 цілком підпорядкована багатопотоковим обчисленням (рисунок 1.2): уніфікований L2 кеш складає 256 кілобайт на кожне ядро, а основний акцент зроблений на кеш-пам'ять третього рівня (8 МБ), що розділяється. Кеш-пам'ять L3 містить усі інструкції і дані з L1 і L2 для зменшення трафіку запитів.

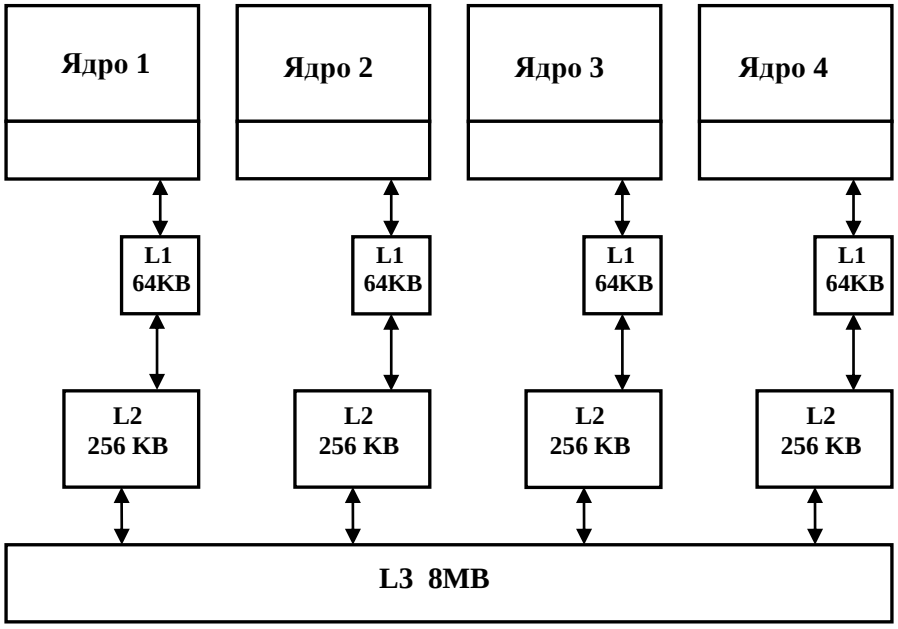


Рисунок 1.2 – Структура Кеш-пам’яті в процессорах Intel Core i7

Замість системної шини FSB використовується високошвидкісний

інтерфейс Quick Path Interconnect (QPI), який забезпечує одночасний обмін даних в обох напрямах (від CPU до північного моста і від північного моста до CPU), що відрізняє його від Front Side Bus. Сумарна пропускна здатність QPI досягає 25,6 GB/s — як мінімум, в 2 рази вище, ніж у FSB.

Багатоядерні AMD процесори. Особливості архітектури чотириядерних процесорів компанії AMD розглянемо на прикладі процесора AMD Phenom II 940 (рисунок 1.3).

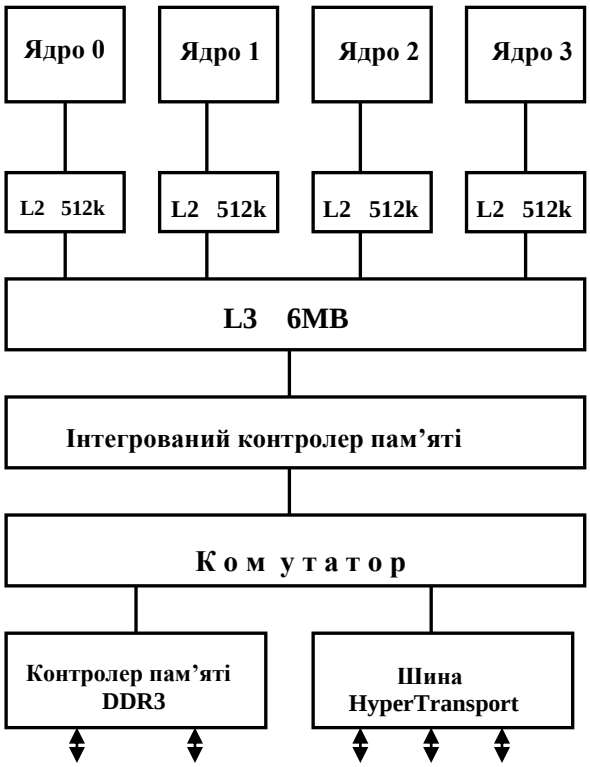


Рисунок 1.3 – Архитектура процессоров AMD Phenom II 940

Процесор Phenom побудований на архітектурі Stars, її нова реалізація містить численні покращення, що дозволяє збільшити число інструкцій, що виконуються за такт. В результаті переходу з 65 на 45-нм техпроцес, напруга живлення, яка потрібна для роботи Phenom II, істотно впала. Разом з покращеннями базової мікроархітектури, це дозволило AMD збільшити тактові частоти. Якщо Phenom першого покоління працювали на частоті 2,6 ГГц, то останні Phenom II починаються від 2,8 ГГц і збільшують частоту до 3,0 ГГц.

Phenom першого покоління не могли бути оснащені великим об'ємом кеш-пам'яті L3 із-за великого теплового пакету при 65-нм технологічному процесі (рисунок 1.4). Струми витоку були дуже сильно понижені при переході на 45-нм

техпроцес, що дозволило AMD збільшити розмір кеш-пам’яті L3 з 2 до 6 Мбайт. Кеш-пам'ять L2 512 кбайт, індивідуальні для кожного ядра, в новому дизайні не змінилися, те ж саме торкається кеш-пам'яті інструкцій і даних рівня L1 по 64 кбайт.

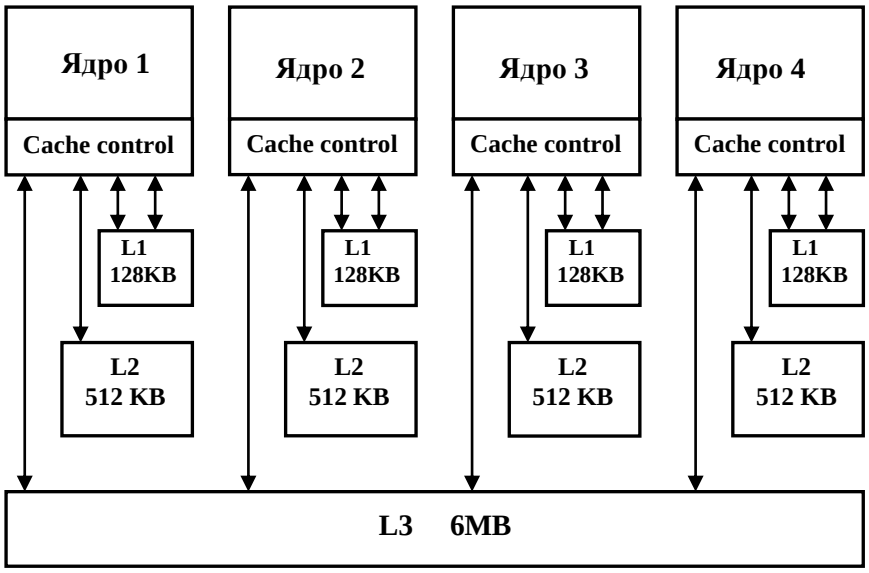


Рисунок 1.4 – Структура кэш памяти в процессоре AMD Phenom II 940

Ще одне покращення продуктивності Phenom II - підтримка пам'яті DDR3.

Багатоядерні SUN процесори. Компанія Sun Microsystems в 2007 році почала випуск високопродуктивного багатоядерного процесора UltraSPARC T2 (рисунок 1.5) [7]. У цьому восьмиядерному процесорі кожне ядро може обробляти до восьми потоків команд. Крім того, цей процесор споживає менше енергії на виконання одного потоку команд в порівнянні з моделями інших виробників. Оскільки кожен потік може забезпечувати роботу окремої операційної системи, такий процесор може використовуватися для створення сервера з 64 віртуальними розділами.

У процесорі UltraSPARC T2 уперше об'єднані такі ключові функціональні можливості, як віртуалізація, обробка даних, мережева підтримка, безпека, обчислення з плаваючою комою і прискорений доступ до пам'яті. За рахунок поєднання усіх цих елементів в одному процесорі скорочується вартість, підвищуються продуктивність, надійність і знижується енергоспоживання. Процесори використовуються для найрізноманітніших завдань, від мережевого

устаткування до високопродуктивних обчислювальних пристроїв або систем зберігання даних.

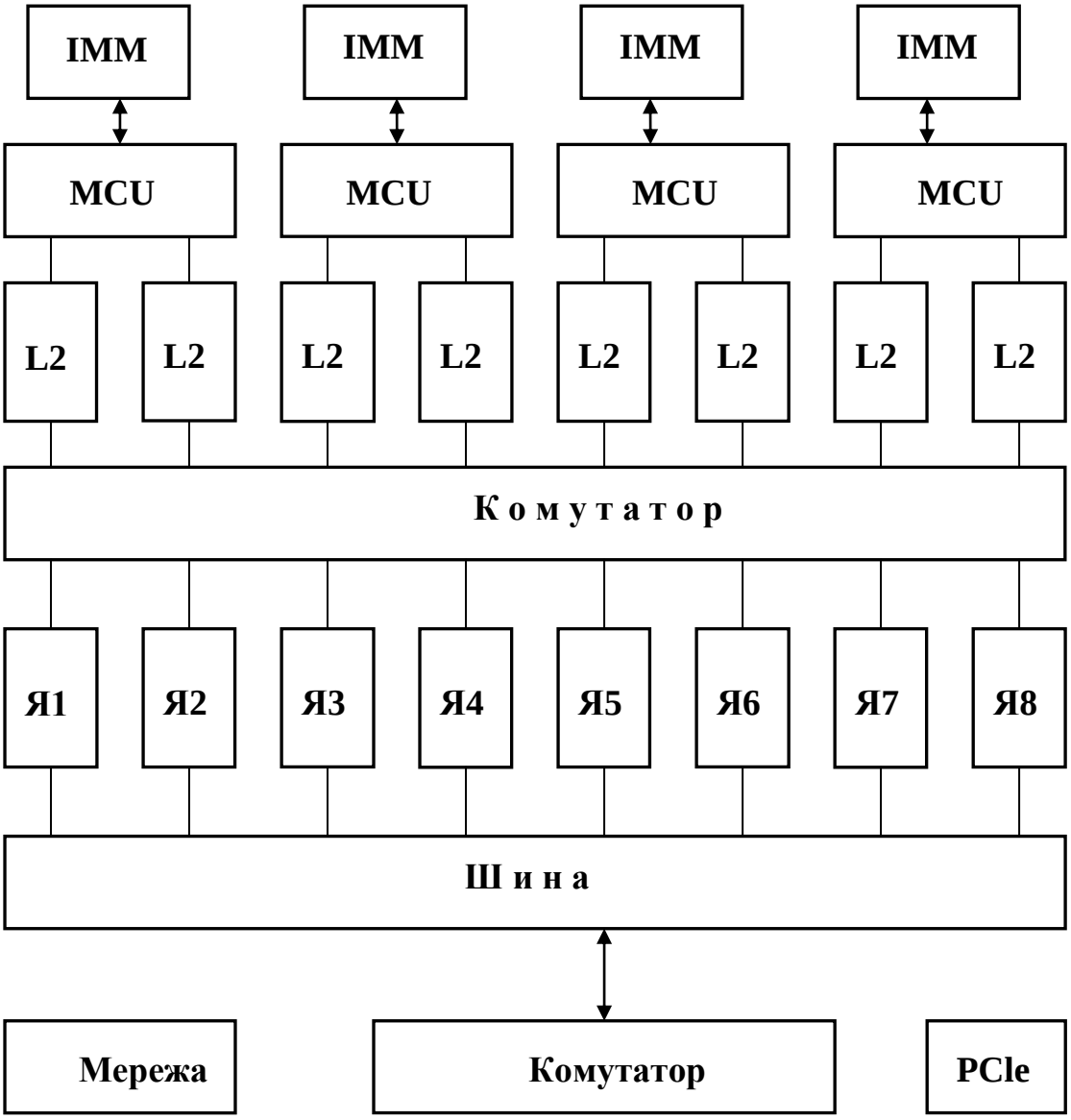


Рисунок 1.5 – Структура процессора SUN ULTRA SPARC T2

Процесор UltraSPARC T2 також може служити і як процесор загального призначення в системах під управлінням багатопотокової операційної системи Solaris з відкритим початковим кодом і інших операційних систем, працюючих в реальному часі, а також нових версій дистрибутива Ubuntu Linux, що відкриває додаткові можливості для розробників. Вісім ядер і підтримку до восьми потоків на кожне ядро забезпечують значне підвищення продуктивності, що було продемонстровано в двох тестах SPEC CPU. Число потоків в процесорі UltraSPARC T2 в два рази перевищує число потоків в процесорі UltraSPARC T1, який нещодавно встановив світовий рекорд в системі з десяти модулів Sun Blade

T6300, що показала результат в 8253,21 операцій в секунду за стандартних умов в тісті SPECjAppServer2004.

Два порти Ethernet (10 гігабіт в секунду) з підтримкою віртуалізації і вбудованою класифікацією пакетів забезпечують швидкий доступ до мереж і взаємодія між серверами.

Організація обчислювальних процесів в КСБА повинна забезпечити реалізацію паралелізму на обох рівнях кластерної систем. На першому рівні - забезпечити ефективну взаємодію вузлів за рахунок мінімальних втрат часу, пов'язаних з передачею даних між вузлами. На другому рівні - організацію обчислювальних процесів у вузлі, пов'язану з обчисленнями в кожному ядрі процесора і взаємодією процесів безпосередньо у вузлі системи.

Для розробки програмного забезпечення для КСБА необхідно використовувати спеціальні програмні засоби, які сьогодні вбудовані безпосередньо в мови програмування або представлені спеціальними бібліотеками [4, 8–12].

Основою реалізації паралельних обчислень в КСБА є концепція процесу (легких потоків). Програмування процесів (потоків) може бути виконане різними способами.

При цьому користувачеві має бути представлена можливість створення процесу або групи процесів, установки пріоритету процесу, місця розміщення в пам'яті і конкретному процесорі (ядрі), блокування і розблокування процесу, завершення процесу. В мовах паралельного програмування (Java, C#, Ада) для цього використовуються спеціальні модулі (класи), які забезпечують необхідні дії із створення процесів [12–16]. Такі модулі повністю описують поведінку процесу за допомогою програмування методу *run()* в Java або тіла задачного модуля в Аді.

Використання так званих потокових функцій, дозволяє заздалегідь описати поведінку процесу у вигляді функції, яка потім як параметр передається спеціальній функції створення процесу або конструктору класу. Реалізовано в мові C# і бібліотеці Win32 [12].

Бібліотеки MPI і PVM реалізують створення процесів у вигляді задач [11,13,17,18]. У MPI використовується поняття комунікатора, який дозволяє

об'єднати процеси в групу, ідентифікувати процеси в групі і організувати колективну взаємодію задач.

OpenMP грунтується на виділенні в послідовній програмі паралельних ділянок за допомогою спеціальних конструкцій [9]. Реалізація паралельних ділянок здійснюється за допомогою ниток (легких процесів). Це спрощує перетворення послідовної програми в паралельну, але не завжди призводить до ефективного варіанту реалізації паралельного алгоритму.

Основою проблемою при розробці програмного забезпечення, заснованого на використанні механізму процесів, являється організація взаємодії процесів. Взаємодія процесів (process interaction) має різноманітні форми, але може бути зведене до двох видів взаємодії :

- передача даних між процесами (process communication)
- синхронізації процесів (process synchronization).

Коректна поведінка паралельної програми критично залежить від організації взаємодії процесів.

Передача даних пов'язана з пересилкою інформації між процесами. При цьому процеси можуть передавати дані з одного процесу в інший або обмінюватися даними. Синхронізація - узгодження процесами своєї поведінки, яка зводиться до того, що процеси обмінюються сигналами для блокування і розблокування свого стану. Обидва види взаємодії пов'язано між собою, оскільки деякі форми передачі даних вимагають синхронізації, а синхронізація може бути пов'язаною з передачею даних.

Взаємодія процесів грунтується на двох моделях паралельного програмування [12]:

- модель, що базується на загальних змінних;
- модель, що базується на передачі повідомлень.

Вид взаємодії процесів залежить від архітектури комп'ютерної системи. У КСБА на рівні взаємодії вузлів використовується модель передачі повідомлень. Всередині вузла вибір моделі взаємодії визначається організацією фізичного зв'язку ядер процесора. На сьогодні зв'язок ядер реалізований через загальну пам'ять, для якої використовується модель взаємодії, заснована на загальних

змінних. У наступних поколіннях багатоядерних процесорів передбачається перехід до розподіленої архітектури, де ядра будуть пов'язані спеціальними лініями зв'язку. У [19] для 8-и ядерних процесорів оптимальною вважається гратчаста топологія, яка забезпечить швидку взаємодію ядер, а також надійність зв'язку ядер. В цьому випадку для взаємодії процесів необхідно використовувати модель, засновану на передачі повідомлень.

Модель, заснована на загальних змінних. Загальні (що розділяються) змінні - це змінні, доступ до яких можливий з кожного процесу, що становить паралельну програму. Як правило, в програмі такі змінні описуються у вигляді глобальних змінних. Розрізняють дві операції над загальними змінними: читання і запис. Взаємодія процесів, що базується на моделі загальних змінних, пов'язана з використанням загальних змінних.

Використання загальних змінних призводить до необхідності рішення двох фундаментальних задач паралельного програмування - задача взаємного виключення і задача синхронізації. Вирішення цих задач здійснюється за допомогою спеціальних програмних засобів, вбудованих в мови програмування, або бібліотек паралельного програмування.

Модель, заснована на передачі повідомлень. Ця модель застосована до різної архітектури. Види моделі передачі повідомлень визначаються реалізацією механізмів і організацією середовища для взаємодії процесів [20]. Механізм взаємодії визначається режимом і характером передачі. Режим задає блокуючу і не блокуючу передачу даних. Характер передачі задає синхронну або асинхронну взаємодію. Синхронна взаємодія при передачі повідомлень представлена в механізмі рандеву в моделі Р. Мильнера (CCS - Calculus of Communicating Systems) [21]. Реалізація механізму рандеву виконана в мові Оккам [22]. Середовище взаємодії забезпечує парну (точкову) або колективну взаємодію. Рандеву забезпечує точкову взаємодію для обміну даними між двома процесами. Середовище MPI дозволяє колективну взаємодію, коли один процес пересилає дані декільком процесам або декілька процесів пересилають дані одному процесу. Для цього в MPI введено поняття комунікатора, що дозволяє об'єднувати процеси в групи і організовувати колективну взаємодію всередині групи.

Модель передачі повідомлень дозволяє зняти проблеми загальної пам'яті, пов'язані з погодженим станом даних, і дозволяє організувати оптимальну передачу даних різної структури.

Взаємодія вузлів пов'язана з передачею даних між вузлами і базується на моделі передачі повідомлень [23]. Використання комп'ютерних мережевих технологій визначає особливості реалізації моделі передачі повідомлень в КСБА, які пов'язані з необхідністю ідентифікації кожного вузла КСБА в мережі, що працює з декільком операційними системами, які в загальному випадку можуть бути різними для кожного вузла КС. Спеціальні програмні засоби, що підтримують комунікацію вузлів КСБА, повинні забезпечити ефективну передачу даних між вузлами з врахуванням вказаних особливостей. Такі засоби спочатку були реалізовані у вигляді спеціальних комутаційних бібліотек, а пізніше стали вбудовуватися безпосередньо в мови програмування [24].

При організації взаємодії вузлів на програмному рівні розрізняють декілька рівнів. Перший низькорівневий забезпечує механізм сокетів [25]. Наступний рівень зв'язується з механізмом віддалених процедур (RPC - Remote Procedure Call) [26]. Віддалені об'єкти, реалізовані в системах CORBA, COM/DCOM забезпечують найбільш високорівневий механізм комунікації вузлів на логічному рівні [3]. Системи MPI (Message Passing Interface) і PVM (Parallel Vitrual Machine) забезпечують ефективну бібліотечну реалізацію механізмів взаємодії вузлів за допомогою моделі передачі повідомлень [11].

Сокети. Реалізовані у бібліотеках Win32, мовах програмування Java, C# [27].

Віддалені процедури. Реалізовані в мовах програмування Java (RMI - Remote Method Invocation), C# (Remoting), Ада (RPC - Remote Procedure Call) [4].

Бібліотека MPI. Взаємодія у бібліотеці MPI базується на використанні функцій Send() і Receive(). Бібліотека підтримує декілька форм цих функцій, забезпечуючи тим самим різноманітні види взаємодії завдань.

Організація взаємодії процесів всередині вузла КСБА визначається архітектурою багатоядерного процесора. Сучасні багатоядерні процесори побудовані на використанні обміну даними за допомогою загальної пам'яті. Тому

основною моделлю взаємодії процесів в ядрі є модель, заснована на загальних змінних.

Загальні змінні - об'єкти, доступ до яких може бути здійснений декількома процесами. Передача даних при використанні загальних змінних пов'язана із записом і зчитуванням даних із загальних змінних. Використання загальних змінних пов'язане з необхідністю вирішення завдань взаємного виключення і синхронізації.

1.3 Моделі паралельних обчислень

Розробка програмного забезпечення для паралельних і розподілених систем має ряд особливостей, головна з яких пов'язана з необхідністю врахування архітектури системи, яка ускладнюється при використанні великого числа процесорів в паралельних системах і рості вузлів в розподілених системах. Крім того, на розробку програми робить істотний вплив і вибрана модель програмування, яка у свою чергу пов'язана з архітектурою системи.

Інтерфейсом між архітектурою і моделлю паралельного програмування може служити модель паралельних обчислень (рисунок 1.6). Її призначення - відображення особливостей архітектури комп'ютерної системи і облік вибраної моделі паралельного програмування. Модель обчислень для розподілених систем повинна відображати взаємодію процесів, яка представляється в моделях програмування [28, 29]. Також модель обчислень дозволяє оцінити наскільки ефективна реалізація програми на вибраній архітектурі. При цьому ефективність визначається за такими критеріями як час виконання програми, завантаження процесорів, передача даних між процесами, наявність тупикових ситуацій та ін.

Крім того, математична модель паралельних обчислень повинна забезпечити ефективну реалізацію програми, що базується на моделі обчислень і на моделі паралельного програмування. Враховуючи складність програм для паралельних і розподілених систем, модель паралельних обчислень повинна дозволити виконати попередній аналіз коректності проведення паралельних процесів і в першу чергу - організації їх взаємодії, що є причиною тупикових

ситуацій.

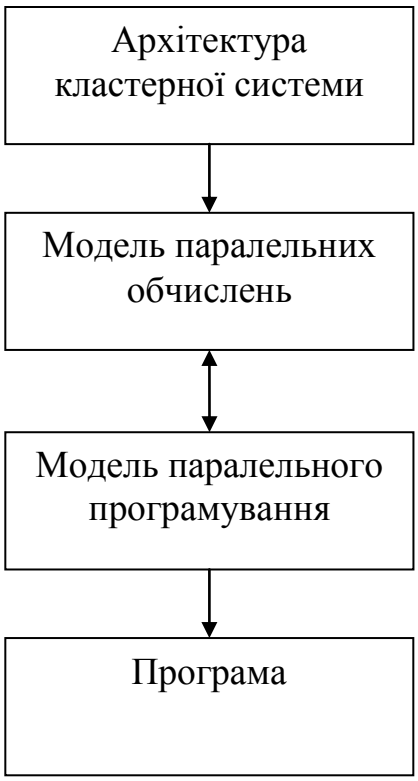


Рисунок 1.6 – Структура взаємозв’язків моделей

Існує досить велика кількість моделей, що дозволяють описати обчислення в паралельних системах (моделі паралельних обчислень). Це моделі Р. Мильнера (*CSS* модель), Ч.Хоара (*CSP* модель), мережі Петрі, мережі на основі кінцевих автоматів та ін. [30]. Різноманітність модель пов'язана з різною спрямованістю цих моделей.

Можливим способом моделювання паралельних обчислень є використання математичних моделей і методів дослідження дискретних систем, розроблених у рамках теорії мереж Петрі [30]. При такому підході в якості моделі програми може бути використана мережа Петрі, що представляється розміченим орієнтованим графом.

При використанні мереж Петрі для опису паралельних програм переходи зазвичай відповідають діям (процесам), а місця - умовам (виділенню або звільненню ресурсів). Розмітка місць в цьому випадку інтерпретується як кількість наявних (нерозподілених) одиниць ресурсу.

У рамках теорії мереж Петрі розроблені методи, що дозволяють для довільної мережі визначити, чи є мережа обмеженою або живою, перевірити

досяжність будь-якого переходу або розмітки мережі. Як результат, ці методи дозволяють визначити наявність безвиході в мережі.

Мережі Петрі в першу чергу спрямовані на моделювання процесів, а не на реалізацію алгоритмів. При цьому у мережах Петрі відсутнє строге поняття процесу для виконання на процесорі. Крім того, реалізація моделі обчислень, побудованої на мережах Петрі, може викликати певні складнощі для користувача.

Останнім часом для опису паралельних обчислень отримали розвиток автоматні моделі [31]. Операції алгебри на множині автоматів дозволяють впровадити нові методи аналізу і синтезу паралельних алгоритмів. Ці методи відпрацьовані в практиці проектування цифрових систем. Крім того, проектування на базі автоматів паралельних систем аж до етапу реалізації може бути єдиним як для програмних, так і для апаратних систем.

Автоматні моделі відрізняються від мереж Петрі більшою потужністю, простотою розуміння, структурною ясністю, великою і перевіреною часом теорією. Їх реалізація зажадає від моделі природності в описі і високій ефективності наступного виконання. Потрібна також наявність формальних процедур аналізу створених алгоритмів.

Алгебра процесів Ч.Хоара. Модель *CSP* (Communicating Sequential Processes) була запропонована Ч.Хоаром [21, 32]. У теорії Ч.Хоара поведінку будь-якого об'єкту можна описати через процес - математичну абстракцію взаємодії системи і її оточення. Кожен об'єкт характеризується набором станів (подій), який задається через алфавіт об'єкту.

Алфавіт - множина імен подій, вибраних для опису конкретного об'єкту. Для кожного об'єкту існує свій, заздалегідь вибраний алфавіт, який залежить від властивостей об'єкту. Запропонована нотація процесу грунтується на тому, що конкретна подія відбувається миттєво (елементарна дія), тобто не має протяжності в часі. Протяжність розглядається як пара подій, які визначають початок і завершення дії. Тривалість дії - інтервал часу між настанням події - початок і настанням події - завершення. Також характерною особливістю даної нотації процесу є абстрагування від прив'язки подій до часу.

Процес означає поведінку об'єкту і може бути описаний в термінах

обмеженого набору подій, вибраних в якості його алфавіту.

Якщо x - деяка подія, а P - процес, то описати об'єкт, який спочатку бере участь в події x, а потім поводиться точно як процес P можна таким чином: $a(x \rightarrow P)=aP$, якщо $x \subseteq aP$.

Будь-який процес може бути представлений у вигляді: ($x: B \rightarrow \mathrm{F(x)}$), де функція ставить у відповідність множині символів x множину процесів B.

Процес, поведінка якого задається рекурсивним співвідношенням, може бути представлений як: ($x: B \rightarrow \mathrm{F}(x, yX.(x: B \rightarrow F(x, \mathrm{X}))$)).

Запропонована нотація дозволяє розглядати будь-який процес як функцію F з областю визначення B, яка виділяє множину подій, які служать як початкові події процесу.

Поведінка об'єкту (процес) – це послідовна зміна станів, яка задається операцією префіксації (→). При описі процесу допускаються також рекурсивність і вибір. Обмежений набір операцій зміни станів не дозволяє описати поведінку складних об'єктів. Проте якщо виходити з того, що КСБА орієнтовані на складні обчислювальні задачі, засновані на крупнозернистому паралелізмі, то моделі Хоара досить для того, щоб промоделювати обчислення в таких системах. Хоча саме недостатність засобів для опису поведінки об'єкту виділяють як основний недолік CSP теорії .

Таким чином, моделлю паралельних обчислень в теорії Ч.Хоара є паралельна поведінка набору об'єктів: ($P_1 \,||\, P_2 \,||\, \ldots \,||\, P_n$), а поведінка i-го об'єкту системи як процес P_i з послідовною зміною станів C_i: αP_i = ($C_1 \rightarrow C_2 \rightarrow \mathrm{C_4} \rightarrow \mathrm{C_3} \rightarrow C_5 \rightarrow \mathrm{C_6}$) на підставі визначеного для об'єкту P_i алфавіту: αP_i={C_1, C_2, C_3, C_4, C_5, C_6 }.

Взаємодія процесів в CSP теорії грунтується на моделі передачі повідомлень. При цьому в алгебрі процесів Хоара визначено поняття логічного каналу, за допомогою якого один процес може переслати дані іншому процесу.

Наявність розвинених засобів для опису взаємодії процесів виділяється як важлива перевага CSP теорії. При цьому алгебра процесів Хоара представляє розвинений математичний апарат не лише для опису поведінки процесів і їх

взаємодії, але дозволяє виконати з його допомогою аналіз коректності самої моделі і, зокрема, представляє інструмент для виявлення тупикових ситуацій ще на етапі проектування програмного забезпечення.

Ще один аргумент на користь застосування CSP теорії для опису моделей паралельних обчислень для КСБА базується на тому, що концепція послідовних взаємодіючих процесів лежить в основі більшості сучасних мов паралельного програмування [4]. Це дозволить спростити реалізацію математичної моделі за допомогою вибору відповідної мови паралельного програмування.

Таким чином, якщо за допомогою CSP теорії можлива побудова математичної моделі обчислень для КСБА з різною структурною організацією, то розробникові програмного забезпечення для КСБА стане доступним ефективний механізм створення програмного забезпечення, що дозволяє скоротити час проектування програмного продукту і підвищити його надійність.

1.4 Постановка задачі дослідження

Розвиток сучасних кластерних систем пов'язаний зі збільшенням обчислювальної потужності вузлів системи, яке на сьогодні можна реалізувати шляхом використання у вузлах багатоядерних процесорів. Розвиток багатоядерних процесорів спрямований на збільшення кількість ядер і вдосконалення системи зв'язків ядер шляхом переходу на розподілену архітектуру, використанням кеш-пам'яті третього рівня, високошвидкісних шин взаємодії підсистем процесора, комутаційних систем.

КСБА характеризуються дворівневим паралелізмом, який визначає паралелізм вузлів системи і паралельну обробку всередині кожного вузла, яка в КСБА набуває важливого значення, оскільки визначає обчислювальну потужність всієї системи.

Організація обчислень в КСБА пов'язана з вирішенням двох задач - організації ефективної взаємодії вузлів системи і обчислень в кожному вузлі, які у свою чергу вимагають ефективної реалізації обчислень в кожному ядрі і їх взаємодії.

Розробка програмного забезпечення для КСБА грунтується на використанні механізму потоків (легких процесів) для програмування обчислень у вузлах і спеціальних механізмах організації взаємодії процесів (реалізовані в мовах програмування Java, C#, Ada і бібліотеках Win32), а також спеціальних засобів, що забезпечують організацію програмування взаємодії вузлів (сокети, віддалені методи, віддалені об'єкти).

КСБА є складною системою, тому організація обчислювальних процесів і розробка програмного забезпечення для неї повинні базуватися на використанні математичних моделей паралельних обчислень, що забезпечують інтерфейс між архітектурою КСБА і моделлю програмування.

У зв'язку з цим актуальною є розробка і вдосконалення моделей паралельних обчислень, що відображають особливості архітектури КСБА, які дозволяють скоротити час розробки програмного забезпечення для КСБА і підвищити його якість.

Метою даної дипломної роботи є вдосконалення організації обчислювальних процесів в кластерних системах з багатоядерною архітектурою, що дозволяє повною мірою реалізувати усі переваги структурної організації КСБА.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– аналіз апаратного забезпечення кластерних систем та визначення основних чинників впливу на організацію обчислень в КСБА;

– аналіз особливостей організації обчислювальних процесів в КСБА, пов'язаних з ускладненням структурної організації системи;

– дослідження моделей паралельних обчислень в КСБА;

– розробка моделей обчислень, що дозволяють описати поведінку і взаємодію процесів в КСБА на рівні вузлів системи при різній організації системи зв'язків вузлів;

– розробка моделей обчислень, що дозволяють описати поведінку і взаємодію процесів у вузлах КСБА, що враховують багатоядерну архітектуру вузла;

– реалізація моделей обчислень в кластерних системах з багатоядерною

архітектурою;

– дослідження тупикових ситуації при використанні об'єктів комунікації і синхронізації в КСБА.

Отже, в даному розділі дипломної роботи виконаний аналіз особливостей організації обчислювальних процесів в КСБА. Показано, що система володіє двома рівнями паралельної обробки – між вузлами і на рівні кожного вузла і вимагає розробки моделей паралельних обчислень, що забезпечують опис поведінки процесів на обох рівнях. Також здійснена постановка задачі дослідження.

2 МАТЕМАТИЧНІ МОДЕЛІ ОРГАНІЗАЦІЇ ОБЧИСЛЕНЬ В КЛАСТЕРНИХ СИСТЕМАХ З БАГАТОЯДЕРНОЮ АРХІТЕКТУРОЮ

2.1 Організація взаємодії вузлів в кластерних системах з багатоядерною архітектурою

Перший рівень КСБА – набір вузлів, що взаємодіють між собою шляхом передачі повідомлень між вузлами. Вузли КС поділяються на вузли для обробки даних і на вузли для зберігання даних. При цьому вузол для обробки дозволяє також і зберігання даних. У загальному випадку КСБА включає один або декілька вузлів обробки і декілька (чи жодного) вузлів для зберігання даних. Вимоги до зв'язку вузлів грунтується на тому, що вузол зберігання пов'язаний, принаймні, з одним вузлом обробки. У загальному випадку дані, розміщені на вузлі зберігання можуть бути доступними більше, ніж для одного вузла обробки.

Кластерна (розподілена) програма - це набір розділів, які можуть виконуватися незалежно і розміщуватися на вузлах КС. Розділи поділяються на активні і пасивні. Активні розділи мають процес, який визначає його поведінку. Пасивний розділ не має нитки управління, але його ресурси (дані, підпрограми) доступні для будь-якого активного розділу.

Оскільки фізичною основою взаємодії розділів є комп'ютерна мережа, то з точки зору можливостей передачі даних кластерна система є повнозв'язною, тобто передача даних з будь-якого вузла у будь-якій іншій здійснюється за один крок без використання проміжних вузлів. Для систем з великим числом вузлів використовується кластеризація, відповідно до якої уся система розбивається на окремі підкластери. Вузли кожного підкластера пов'язані одним комутатором, а уся система в цьому випадку розглядається як набір підкластерів, які фізично зв'язуються через з'єднання комутаторів. В порівнянні з централізованою структурою КС, де використовується один центральний комутатор, кластеризація КС дозволяє розвантажити як центральний комутатор, так і локальні комутатори. Це досягається тим, при розробці схеми взаємодії вузлів основна частина обмінів планується всередині кожного підкластера з обмеженим виходом на центральний

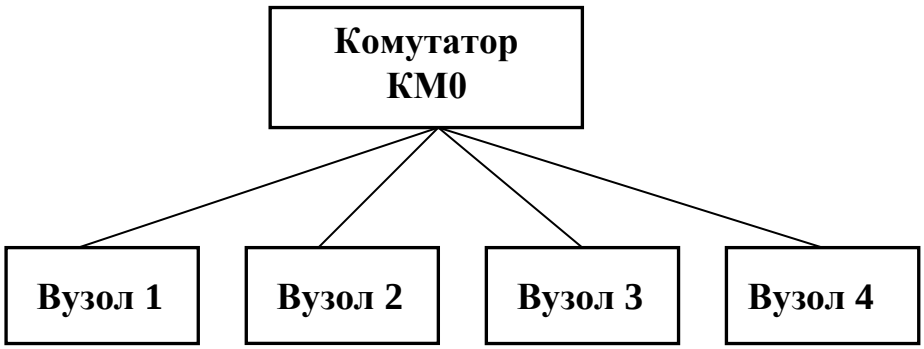
комутатор. На рисунку 2.1 представлені можливі схеми з'єднання вузлів КС. На рисунку 2.1 а) використовується один комутатор (КМ1). При великому потоці даних між вузлами 1-4 комутатор отримуватиме велике навантаження і може стати причиною затримок при передачі даних. На рисунку 2.1 б) вузли 1 і 2 об'єднані в підкластер 1 і пов'язані окремим комутатором (КМ1). Вузли 3 і 4 об'єднані в підкластер 2 і пов'язані окремим комутатором (КМ2). Передача даних між підкластерами 1 і 2 здійснюється через пряме зв'язування комутаторів КМ1 і КМ2. На рисунку 2.1 в) для зв'язування підкластерів використовується додатковий комутатор КМ0.

На логічному рівні взаємодія розділів грунтується на моделі передачі повідомлень. Традиційно розділи діляться на передавальні і приймаючі. Передавальний розділ формує дані і передає приймаючому розділу. Проте може мати місце взаємодія, під час якої розділи обмінюються даними, тобто, немає жорсткого ділення на того, що приймає і передає розділи. В цьому випадку розділи можна підрозділяти на того, що викликає і викликається. При цьому розділ, що викликає ініціює взаємодію, розділ, що викликається, - чекає виклику. При встановленні зв'язку між розділами відбувається передача інформації у будь-якому напрямі:

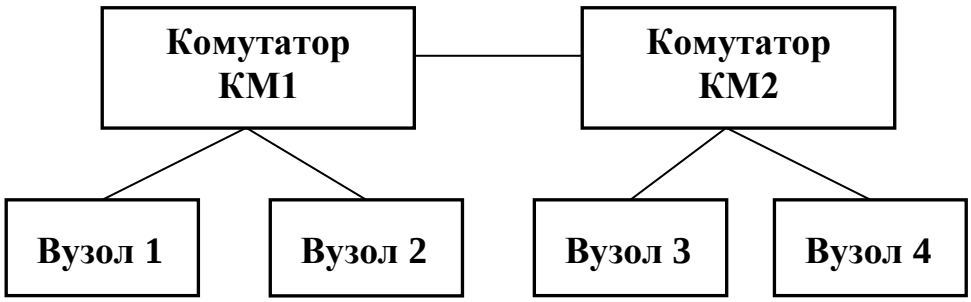
- від розділу, що викликає до розділу, що викликається,
- від розділу, що викликається, до того, що викликає,
- обмін даними між розділами.

Існують різні реалізації моделі передачі повідомлень, засновані на низькорівневих засобах (сокетах), віддалених процедурах (RPC механізмах), віддалених об'єктах (CORBA, COM/DCOM) [33].

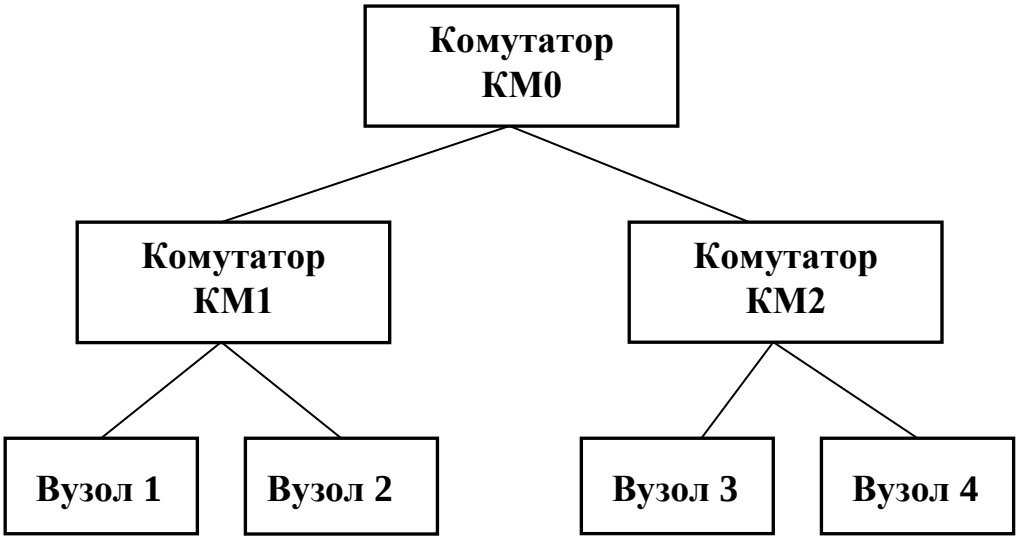
Сокети забезпечують однонапрямлену передачу даних між розділами. Віддалені процедури забезпечують двонаправлену передачу даних при використанні вхідних (in), вихідних (out) або універсальних (in out) параметрів віддаленої процедури.



а) один комутатор



б) два комутатори



в) три комутатори

Рисунок 2.1 – Схеми комутації вузлів кластерної системи

2.2 Розробка моделі обчислень на рівні вузлів системи

Основне призначення моделі обчислень першого рівня КСБА1 - опис поведінки кожного вузла і організації взаємодії вузлів КСБА.

При розробці моделі обчислень КСБА1 представимо, відповідно до алгебри процесів Ч.Хоара, систему вузлів як набір об'єктів. При цьому розділимо вузли на серверні і клієнтські і, відповідно, виділимо об'єкт сервер і множину об'єктів клієнтів (рисунок 2.2).

Набір подій, пов'язаних з об'єктом сервер (алфавіт сервера) *С* представимо у вигляді:

$$\alpha C = \{ C.\ввід, C.\вивід, C.\прийняти.K_i, C.\передати.K_i\ C.\обчислення \} \quad (2.1)$$

де події:

- *С. ввід* - ввід даних на сервері
- *С. вивід* - вивід результату на сервері
- *С. прийняти.*K_i - прийом сервером даних від клієнта K_i
- *С. передати.*K_i - передача даних від сервера клієнтові K_i,
- *С. обчислення* - обчислення на сервері.

Набір подій, пов'язаних з *і*-м об'єктом клієнтом (K_i) залежить від можливої системи зв'язків сервера і клієнтів. На рисунку 2.2 представлена схема взаємодії (схема 1), де об'єкти клієнти взаємодіють тільки з одним північним об'єктом. Тут неможлива пряма взаємодія клієнтів. Для передачі даних між клієнтами необхідно задіяти об'єкт сервер.

На рисунку 2.3 представлена схема взаємодії (схема 2), де об'єкти клієнти можуть взаємодіяти як з північним об'єктом, так і безпосередньо між собою.

Для схеми 1 алфавіт клієнта можна представити у вигляді:

$$\alpha Ki = \{ K_i.прийняти.C, K_i.передати.C, K_i.обчислення\} \quad (2.2)$$

де події:

- K_i*.прийняти.С* - прийом даних від сервера

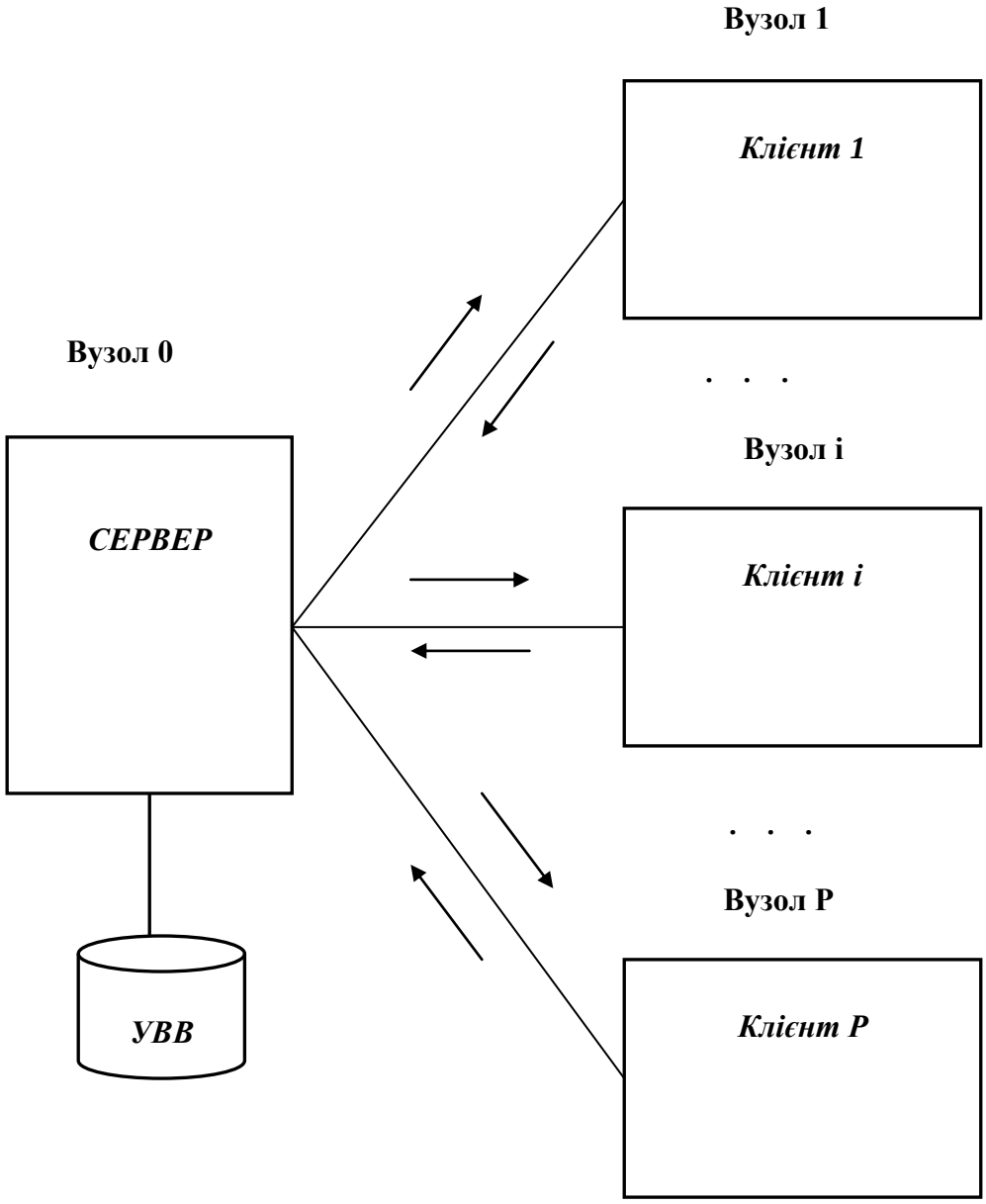


Рисунок 2.2 – Схема 1 взаємодії вузлів КСБА

– *$К_i$.передати.С - передача даних серверу,*

– *$К_i$.обчислення - обчислення на клієнтському об'єкті.*

Поведінку об'єкту сервера використовуючи алфавіт сервера *αС,* можна описати як послідовний процес

СЗ = { С. ввід → С. передати. $К_i$ → С. обчислення → С. прийняти.$К_i$ → С. вивід }

Для множини об'єктів клієнтів $К_i$ їх поведінка (клієнтський процес) описується таким чином:

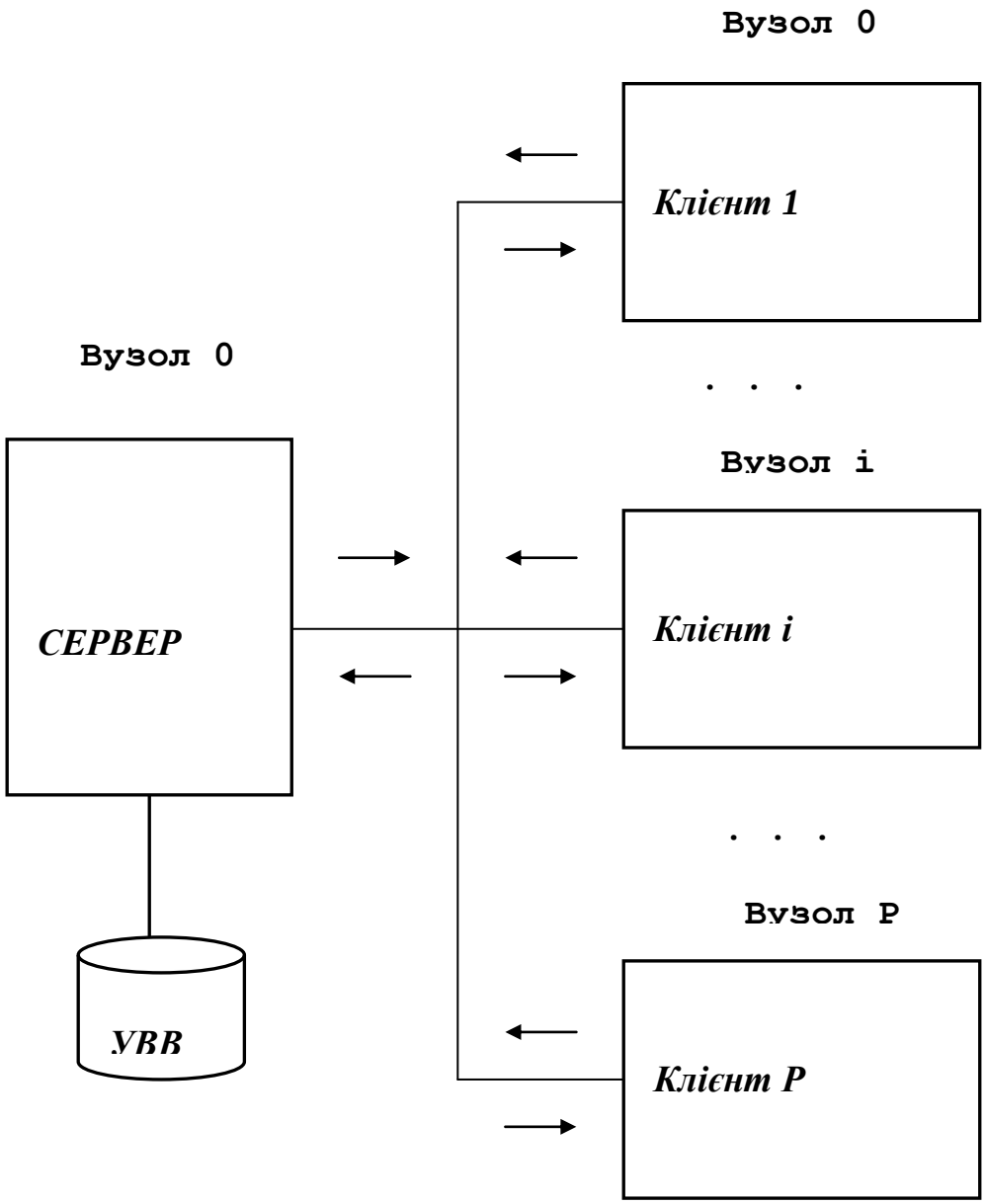


Рисунок 2.3 – Схема 2 взаємодії вузлів КСБА

K_i ={ K_i.*прийняти.С* → K_i .*обчислення* → $_{Ki}$.*передати.С*

Для схеми 2 алфавіт клієнта можна представити у вигляді

αK_i = { *К.прийняти.С,* K_i .*передати.С,* K_i .*обчислення*

K_i .*прийняти.*K_j, K_i .*передати.*K_j }

де події:

- K_i.*прийняти.С, - прийом даних від сервера*
- K_i .*передати.С - передача даних серверу,*
- K_i .*обчислення - обчислення на клієнтові,*
- K_i .*прийняти.*K_j *- прийом даних від клієнта* K_j,
- K_i .*передати.*K_j *- передача даних клієнтові* K_j .

Поведінка об'єкту сервера на схемі 2, використовуючи алфавіт сервера *αС,* можна описати як процес

C = (*С. ввід*$\rightarrow$ *С. передати.* $К_i \rightarrow$ *С. обчислення* $\rightarrow$ *С. прийняти.*$К_i \rightarrow$ *С. вивід)*

Для множини об'єктів клієнтів *Кі* їх поведінка (клієнтський процес) описується таким чином

$К_i$ = ($К_i$ *.прийняти.С* $\rightarrow$ $К_i$ *.передати.* $К_i \rightarrow$ $К_i$ *.обчислення* $\rightarrow$

$К_i$ *.прийняти.*$К_j \rightarrow$ *К.передати.С*)

Поведінка моделі КСБА1 описується як паралельна комбінація поведінки процесів *С* і $К_i$:

КЛІЄНТИ = ($К_1$|| $К_2$|| . . .|| $К_P$), *СЕРВЕР* = (*С*)

КСБА1 = (*СЕРВЕР* || *КЛІЄНТИ*)

Найважливішою складовою поведінки об'єктів в моделі КСБА1 є їх взаємодія. Ці дії для об'єктів *А* і *В* закладені в події *А.прийняти.В* і *А.передати.В,* де об'єкт *А* приймає (передає) дані від об'єкту *В.*

Взаємодія вузлів в моделі КСБА1 здійснюється за допомогою передачі повідомлень між: а) серверними і клієнтськими об'єктами, б) між клієнтськими об'єктами. У CSP теорії взаємодія розглядається як подія, в якій беруть участь два об'єкти і яка пов'язана з передачею повідомлень. Взаємодія процесів здійснюється через канал, куди один процес записує дані, а інший процес - зчитує ці дані. При цьому процеси синхронізують свою поведінку, оскільки взаємодія відбувається тільки тоді, коли обидва процеси звертаються до каналу.

Канал характеризується ім'ям (*КН*) і даними (*Д*), які передаються через канал. Взаємодія при використанні каналу є подією, яка описується парою *КН.Д.* Множина повідомлень, які можна передати в процесі *Q* через канал *КН*, задається таким чином:

$\alpha КН(Q) = \{ Д \mid КН.Д \in \alpha Q \}$

Для отримання інформації, пов'язаної з каналом, визначено дві функції, що дозволяють отримати ім'я каналу і дані, що передаються по каналу: *кнл(КН.Д.) = К, сбщн(КН.Д.) = Д.* Для цього запису використовуються операція запису в канал - *КН ! Д* і операція зчитування з каналу *КН ? Д.*

Якщо процес *Q* використовує канал *z* для передачі даних *e*, то його поведінку можна описати таким чином:

(*z!e* $\rightarrow$ Q) = (*z.e* $\rightarrow$ Q)

При цьому після запису в канал *z* процес поводиться як *Q*.

Для процесу *Q*, який в початковому стані готовий до читання даних *t* з каналу *s*, а потім поводиться як *Q(t)* має місце поведінка, що описується як

(*s?t* $\rightarrow$ Q(*x:* { *x* | *кнл(x)* = s } $\rightarrow$ *Q(сбщн(x)*))

Для організації коректної взаємодії двох процесів *A* і *B* через канал *S* потрібний збіг алфавітів *αs(A) = αs(B)*.

Поняття каналу, що використовується в теорії Ч.Хоара, має недоліки, пов'язані з тим, що канал дозволяє передавати дані тільки в одному напрямі, канал забезпечує тільки синхронну передачу даних, канал підтримує тільки один формат даних для передачі. Це не дозволяє повною мірою описати різні види взаємодії об'єктів через передачу повідомлень, які мають місце для серверних і клієнтських об'єктів в моделі КСБА1, наприклад, при використанні механізму викликів віддалених процедур.

Введемо для опису взаємодії об'єктів в КСБА, де передбачаються різні види взаємодії (однонапрямлені і двонаправлені), засновані на передачі повідомлень, поняття об'єкту комунікації (*ОК*). Об'єкт комунікації є розширенням поняття канал і є параметризованим об'єктом *ОК*(*Д, Н, В, Б*), де параметри:

Д - визначає тип і об'єм даних, які можуть передаватися через *ОК*;

Н - визначає напрям передачі даних (*in, out*);

В - вид взаємодії (синхронний (*С*) або асинхронний (*А*));

Б - тип взаємодії (буферизований (Б) або небуферизований (*НБ*)).

За допомогою об'єкту комунікації можна описати складніші види передачі даних, які використовуються при організації взаємодії серверних і клієнтських об'єктів в моделі КСБА1.

Алфавіт *ОК* включає наступні стани:

αОК = { *ОК.передати, ОК.прийняти* }

де події:

– *ОК.передати* - передати виклик *ОК*

– *ОК.прийняти* - прийняти виклик *ОК.*

Поведінка об'єкту комунікації *ОК* описується рекурсивно таким чином:

$ОК = ((ОК.прийняти \mid ОК.передати) \rightarrow ОК)$

Відмінність *ОК* від прийнятого в теорії Т. Хоара поняття канал полягає, зокрема, в тому, що об'єкт комунікації може одночасно передавати дані у будь-якому напрямі, тоді як канал використовується для передачі даних тільки в одному напрямі.

Важливою особливістю об'єкту комунікації є те, що процес, що описує поведінку *ОК*, є підлеглим процесом по відношенню до процесів, які його використовують. У теорії Ч.Хоара між процесами *P* і *T* існує відношення підлеглості, якщо $\alpha P \subseteq \alpha T$. При паралельному виконанні процесів *P* || *T* події в процесі *P* можуть наступити тільки тоді, коли це дозволяє процес *T*. При цьому події підмножини $(\alpha T - \alpha P)$ в процесі *T* можуть виконуватися незалежно від процесу *P*. При такій поведінці процесів *P* розглядається як підлеглий процес до *T*, процес Т є головним (основним) процесом і використовується позначення Р // Т. Для підлеглого процесу вводиться ім'я *z*, яке використовується в основному процесі для опису взаємодії з підлеглим: *z* : *T* або (*z* : Р // Т).

Процес, що описує поведінку об'єкту комунікації *ОК* розглядається як підлеглий процес (підпроцес) до процесу *X* і вірне співвідношення *ОК // X* = (*ОК* || *X*) \ *αОК*. При цьому *αОК* $\subseteq$ *αX* і *α(ОК // X*)= (*αОК* – *αX*). Якщо ввести ім'я об'єкту комунікації *m*, то (*m* : *ОК // X*).

Розглядаючи *ОК* як підлеглий процес, можна сформулювати правила щодо алфавітів і поведінки взаємодіючих процесів, які визначають коректність взаємодії процесів через *ОК*.

Перше правило визначає необхідну умову взаємодії двох процесів по відношенню до алфавітів процесів і алфавіту *ОК*.

“Якщо *P* і *Q* - процеси, *Z* - об'єкт комунікації, то необхідною умовою взаємодії об'єктів *P* і *Q* через *Z* є виконання співвідношення $\alpha Z \subseteq (\alpha P \cup \alpha Q)$ “.

Друге правило визначає необхідну умову взаємодії двох процесів по відношенню до поведінки процесів.

“Якщо *P* і *Q* - процеси, *Z* - об'єкт комунікації, то необхідною умовою

взаємодії *P* і *Q* через *Z* є виконання наступних подій, що визначають поведінку цих процесів:

Z = ((Z.прийняти | Z.передати) → Z)

P = ((Z.передати) → P), Q = ((Z.прийняти) → Q)

(Z / / (P | | Q)).

Модель КСБА1 дозволяє аналізувати і виявляти причини виникнення тупиків при використанні об'єктів комунікації. Для об'єкта комунікації двох процесів *P* і *Q* через *ОК* (*Д, Н, В, Б*) причинами виникнення тупикової ситуації можуть бути наступні:

- фактичні параметри для *ОК.Д* в процесах *P* і *Q* не відповідають значенням формальних параметрів у *ОК.Д* при динамічному формуванні значень, що передаються ;

- параметр *ОК.Н* не відповідає напрямку переданої (прийнятої) інформації, яка закладена в об'єкті комунікації;

- використовується синхронна форма (*ОК.В = С*) взаємодії, яка пов'язана з поверненням результату;

- при буферизованій формі (*ОК.Б = В*) взаємодії розмір буфера не відповідає розміру даних, які передаються (приймаються) між процесами.

2.3 Математична модель обчислень у вузлах кластерних систем з багатоядерною архітектурою

Вузол КСБА, реалізований на основі багатоядерного процесора є складним об'єктом, що складається з декількох об'єктів, що виконуються паралельно і взаємодіють між собою. При цьому на фізичному рівні ці об'єкти пов'язані з ядрами багатоядерного процесора, а на логічному – з процесами, що виконуються в ядрах процесора.

Поведінка об'єкту - вузла залежить від структури зв'язків, за допомогою якої організована передача даних між ядрами. У сучасних багатоядерних процесорах використовується два підходи до організації системи зв'язку ядер. Перший підхід грунтується на загальній пам'яті і шинній організації (шинна архітектура), другий

– на використанні прямого зв'язку ядер (розподілена архітектура). На логічному рівні для шинної організації і загальної пам'яті використовується модель передачі повідомлень, для прямої системи зв'язків - модель передачі повідомлень .

Багатоядерні процесори, що існують на сьогодні, використовують шинну організацію. Восьмиядерні процесори планується створювати на основі розподіленої архітектури. Передбачається, що це буде структура типу «решітка», яка покликана забезпечити надійність багатоядерного процесора у разі виходу з ладу ядер або зв'язків. При реалізації моделі другого рівня КСБА2 розглянемо обидві можливі архітектури багатоядерного процесора з числом ядер в процесорі P = 8. В цьому випадку структура зв'язків ядер в ньому може бути реалізована як за допомогою шинної архітектури, так і за допомогою розподіленої архітектури. Для розподіленої архітектури розглянемо топологію типу «решітка».

При розробці моделі паралельних обчислень КСБА2 у вузлі КСБА розглядатимемо вузол як складну систему, що включає набір об'єктів, заснованих на об'єктах двох типів: об'єкт майстер і об'єкт робочий (рисунок 2.4).

Об'єкт майстер виконує дві функції:

- відповідає за зв'язок з рештою вузлів КСБА, і, зокрема, – з серверним вузлом КСБА;

- є сервером всередині вузла.

Об'єкт майстер (*M*) приймає початкові дані від сервера КСБА, розподіляє їх у вузлі по об'єктах робочих (*P*), збирає результати обчислень у вузлі і посилає результат в серверний вузол КСБА. Об'єкт робочий *P* приймає дані від об'єкту майстер, виконує обчислення і повертає результат об'єкту майстер.

Набір подій, пов'язаних з об'єктом майстер *і-го* вузла (M_i), можна представити у вигляді:

$\alpha M_i = \{ M_i.передати.P_{ij}, M_i.обчислення, M_i.принять.P_{ij} \}$

де події:

- $M_i.передати.p_{ij}$, – передати дані у вузлі об'єкту робочий P_{ij}
- M_i *.обчислення* – обчислення в об'єкті майстер
- $M_i.передати.p_{ij}$, – прийняти дані у вузлі від об'єкту робочий P_{ij}.

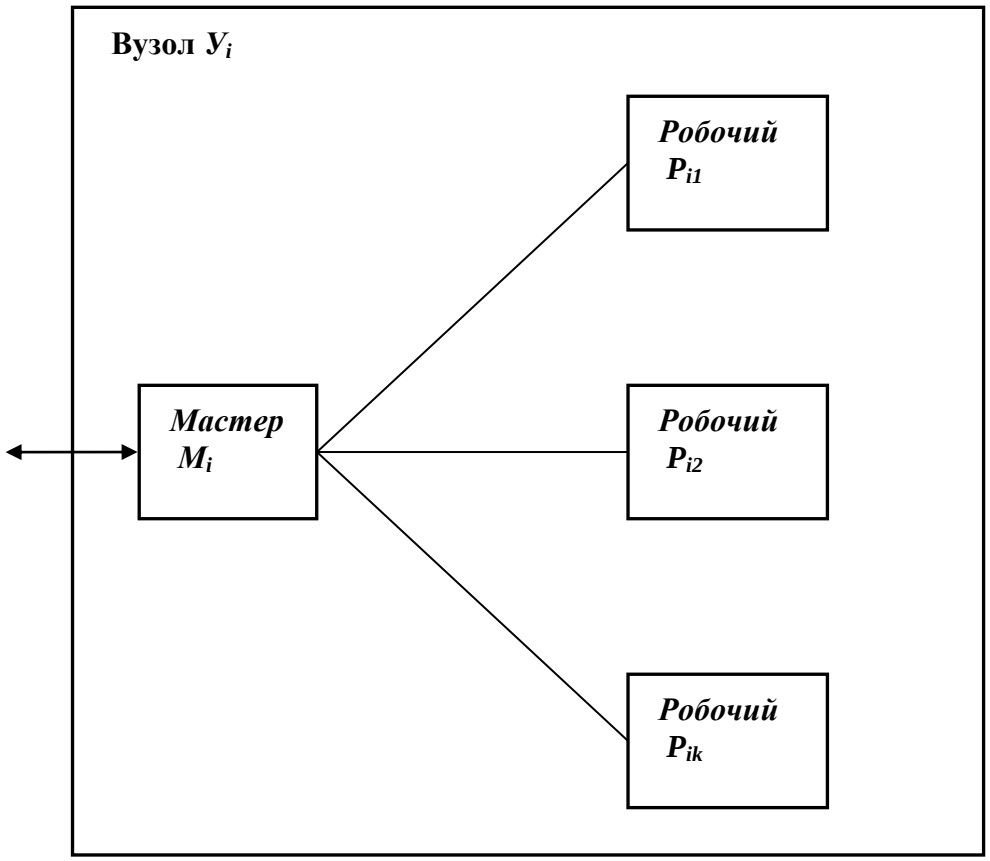


Рисунок 2.4 – Структура об’єкта *Вузол*

Набір подій, пов'язаних з об'єктом *j*-ий робочий всередині *i*-го вузла (P_{ij}) можна представити у вигляді

$$\alpha P_{ij} = \{ P_{ij}.приняти.M_i, P_{ij}.передати.M_i, P_{ij}.обчислення$$

$$P_{ij}.приняти.P_{ik}, P_{ij}.передати.P_{ik} \}$$

де події:

- $P_{ij}.приняти.M_i$ – прийом даних від об'єкту майстер
- $P_{ij}.передати.M_i$ – передача даних об'єкту майстер
- $P_{ij}.обчислення$ – обчислення в об'єкті робочий
- $P_{ij}.приняти.P_{ik}$, – прийом даних від *к*-го об'єкту робочий
- $P_{ij}.передати.P_{ik}$ – передача даних для *к*-го об'єкту робочий.

Опис поведінки об'єктів майстер і робочий залежить від системи зв'язків ядер в багатоядерному процесорі, яка визначатиме форму взаємодії цих об'єктів.

Якщо зв'язок ядер в багатоядерному процесорі реалізований через розподілену архітектуру з допомогою, наприклад топології зірка, де всі ядра зв'язані через центральне ядро, то поведінку об'єктів всередині вузла можна

описати таким чином.

Поведінка хост об'єкту майстер M_i використовуючи алфавіт αM_i, можна описати як

$M_i = (M_i .передати.P_{ij} \rightarrow M_i .обчислення \rightarrow M_i .прийняти.P_{ij})$

Поведінка об'єкту *робочий* P_{ij} :

$P_{ij} = (P_{ij} .прийняти.Mi \rightarrow P_{ij} .обчислення \rightarrow P_{ij} .передати.M_i)$

Модель другого рівня для *i*-го клієнтського вузла можна описати як паралельну комбінацію поведінки процесів M_i і P_{ij}:

$Мастер_i = (M_i)$, $Робочі_i = (P_1 || P_2 || \ldots || P_к)$

$Вузол_i = (M_i || Робочі_i)$

Організація взаємодії процесів в моделі *КСБА2* залежить від вибраної системи зв'язків ядер. Для прямого зв'язку ядер (розподілена архітектура) використовується модель, заснована на передачі повідомлень, для зв'язку через пам'ять (шинна архітектура) – модель, заснована на загальних змінних.

Модель передачі повідомлень. Розглядатимемо вузол як об'єкт, заснований на використанні восьмиядерного процесора, система зв'язків якого реалізована по розподіленій архітектурі.

У вузлі з лінійною архітектурою поведінка *майстер* об'єкту (M_i) описується таким чином:

$M_i = (M_i.передати.p_{i1} \rightarrow M_i.обчислення \rightarrow Mi.прийняти.p_{i1})$

У вузлі з лінійною архітектурою поведінка об'єкту *робочий* описується таким чином:

$P_{i1} = (P_{i1}.прийняти.M_i \rightarrow P_{i1}.передати.P_{i2} \rightarrow P_{i1}.обчислення \rightarrow$

$P_{i1}.прийняти.P_{i2} \rightarrow P_{i1}.передати.M_i)$

$P_{i2} = (P_{i2}.прийняти.P_{i1} \rightarrow P_{i2}.передати.P_{i3} \rightarrow P_{i2}.обчислення \rightarrow$

$P_{i2}.прийняти.P_{i3} \rightarrow P_{i2}.передати.P_{i1})$

$P_{i3} = (P_{i3}.прийняти.P_{i2} \rightarrow P_{i3}.передати.P_{i4} \rightarrow P_{i3}.обчислення \rightarrow$

$P_{i3}.прийняти.P_{i4} \rightarrow P_{i3}.передати.P_{i2})$

$P_{i4} = (P_{i4}.прийняти.P_{i3} \rightarrow P_{i4}.передати.P_{i5} \rightarrow P_{i4}.обчислення \rightarrow$

$P_{i4}.прийняти.P_{i5} \rightarrow P_{i4}.передати.P_{i3})$

$P_{i5} = (P_{i5}.прийняти.P_{i4} \rightarrow P_{i5}.передати.P_{i6} \rightarrow P_{i5}.обчислення \rightarrow$

$P_{i5}.приняти.P_{i6} \rightarrow P_{i5}.передати.P_{i4}$)

P_{i6} = ($P_{i6}.приняти.P_{i5} \rightarrow P_{i6}.передати.P_{i7} \rightarrow P_{i6}.обчислення \rightarrow$

$P_{i6}.приняти.P_{i7} \rightarrow P_{i6}.передати.P_{i5}$)

P_{i7} = ($P_{i7}.приняти.P_{i6} \rightarrow P_{i7}.обчислення.P_{i7} \rightarrow P_{i7}.передати.P_{i6}$)

Події $P_{ij}.прийняти.P_{ik}$, $P_{ij}.передати.P_{ik}$ пов'язані з взаємодією об'єкту *робочий i* з об'єктом *робочий k* у вузлі *j*.

У загальному випадку в лінійній структурі для множини об'єктів *Робочий* $P_{iк}$ їх поведінку можна описати таким чином:

для $k=1$

P_{i1} = ($P_{i1}.приняти.M_i \rightarrow P_{i1}.передати.P_{i2} \rightarrow P_{i1}.обчислення \rightarrow$

$P_{i1}.приняти.P_{i2} \rightarrow P_{i1}.передати.M_i$)

для $1 < k < 7$

$P_{iк}$ = ($P_{iк}.принятb.P_{iк-1} \rightarrow P_{iк}.передатb.P_{iк+1} \rightarrow$

$P_{iк}.обчислення.P_{iк}, \rightarrow P_{iк}.приняти.P_{iк+1} \rightarrow P_{iк}.передати.P_{iк-1}$)

для $k=7$

P_{i7} = ($P_{i7}.приняти.P_{i6} \rightarrow P_{i7}.обчислення \rightarrow P_{i7}.передать.P_{i6}$)

У вузлі з кільцевою топологією поведінка об'єкту майстер описується таким чином:

M_i = ($M_i.передати.p_{i1} \rightarrow M_i.передати.p_{i7} \rightarrow M_i.обчислення \rightarrow$

$M_i.прийняти.p_{i7} \rightarrow M_i.прийняти.p_{i1}$)

У вузлі з кільцевою топологією поведінка об'єкту робочий описується таким чином:

P_{i1} = ($P_{i1}.передати.P_{i2} \rightarrow P_{i1}.обчислення \rightarrow P_{i1}.приняти.P_{i2}$)

P_{i2} = ($P_{i2}.приняти.P_{i1} \rightarrow P_{i2}.передати.P_{i3} \rightarrow P_{i2}.обчислення \rightarrow$

$P_{i2}.приняти.P_{i3} \rightarrow P_{i2}.передати.P_{i1}$)

P_{i3} = ($P_{i2}.приняти.P_{i2} \rightarrow P_{i2}.передати.P_{i4} \rightarrow P_{i2}.обчислення \rightarrow$

$P_{i2}.приняти.P_{i4} \rightarrow P_{i2}.передати.P_{i2}$)

P_{i4} = ($P_{i4}.приняти.P_{i3} \rightarrow P_{i4}.обчислення \rightarrow P_{i4}.передати.P_{i3}$)

P_{i7} = ($P_{i7}.приняти.M_i \rightarrow P_{i7}.передати.P_{i6} \rightarrow P_{i7}.обчислення \rightarrow$

$P_{i7}.приняти.P_{i2} \rightarrow P_{i7}.передати.M_i$)

P_{i6} = ($P_{i6}.приняти.P_{i5} \rightarrow P_{i6}.передати.P_{i5} \rightarrow P_{i6}.обчислення \rightarrow$

$P_{i6}.приняти.P_{i3} \rightarrow P_{i6}.передати.P_{i5}$)

P_{i5} = ($P_i5.приняти.P_{i6} \rightarrow P_{i5}.обчислення \rightarrow P_{i5}.передати.P_{i6}$)

Події $P_{iк}.приняти.P_{im}$, $P_{iк}.передати.P_{im}$ зв’язані *з* взаємодією *k*-го *робочого* з *m*–м *робочим.* Особливістю поведінки майстер об'єкту в кільцевій архітектурі є його зв'язок з двома об'єктами *робочий* (P_1 і P_7). Це дозволяє реалізувати паралельну розсилку даних і збірку результатів по двох гілках: $M \rightarrow P_1 \rightarrow P_2 \rightarrow P_3$ і $M \rightarrow P_7 \rightarrow P_6 \rightarrow P_5 \rightarrow P_4$

У вузлі з топологією зірка поведінка об'єкту *майстер* описується таким чином (рисунок 2.5):

M_i = ($M_i.передати.P_{i1} \rightarrow M_i.передати.P_{i2} \rightarrow$

$M_i.передати.P_{i3} \rightarrow M_i.передати.P_{i4} \rightarrow M_i.передати.P_{i5} \rightarrow$

$M_i.передати.P_{i6} \rightarrow M_i.передать.P_{i7} \rightarrow M_i.обчислення \rightarrow$

$M_i.прийняти.X_iP_{i1} \rightarrow M_i.прийняти.X_iP_{i2} \rightarrow M_i.приняти.X_iP_{i3} \rightarrow$

$M_i.прийняти.X_iP_{i4} \rightarrow M_i.прийняти.X_iP_{i5} \rightarrow M_i.прийняти.X_iP_{i6} \rightarrow$

$M_i.прийняти.P_{i7}$)

У вузлі з топологією зірка поведінка об'єкту *робочий* описується таким чином:

P_{i1} = ($P_{i1}.прийняти.M_i \rightarrow P_{i1}.обчислення \rightarrow P_{i1}.передати.M_i$)

P_{i2} = ($P_{i2}.прийняти.M_i \rightarrow P_{i2}.обчислення \rightarrow P_{i2}.передати.M_i$)

P_{i3} = ($P_{i3}.прийняти.M_i \rightarrow P_{i3}.обчислення \rightarrow P_{i3}.передати.M_i$)

P_{i4} = ($P_{i4}.прийняти.M_i \rightarrow P_{i4}.обчислення \rightarrow P_{i4}.передати.M_i$)

P_{i5} = ($P_{i5}.прийняти.M_i \rightarrow P_{i5}.обчислення \rightarrow P_{i5}.передати.M_i$)

$P_{i\,6}$ = ($P_{i6}.прийняти.M_i \rightarrow P_{i6}.обчислення \rightarrow P_{i6}.передати.M_i$)

P_{i7} = ($P_{i7}.прийняти.M_i \rightarrow P_{i7}.обчислення \rightarrow P_{i7}.передати.M_i$)

Скорочений запис поведінки об'єктів майстер і робітник:

M_i = ($M_{i.}.передати.P_{ij} \rightarrow M_{i.}.прийняти.P_{ij}$)

P_{ij} = (P_{ij} .$приняти.M_{i.} \rightarrow P_{ij}$ $обчислення_j \rightarrow P_{ij}.передати.M_{i.}$)

де j = 1 . . 7.

Події $P_{ij}.прийняти.M_i$, $P_{ij}.передати.M_i$ пов'язані з взаємодією *j*-го робітника з майстром.

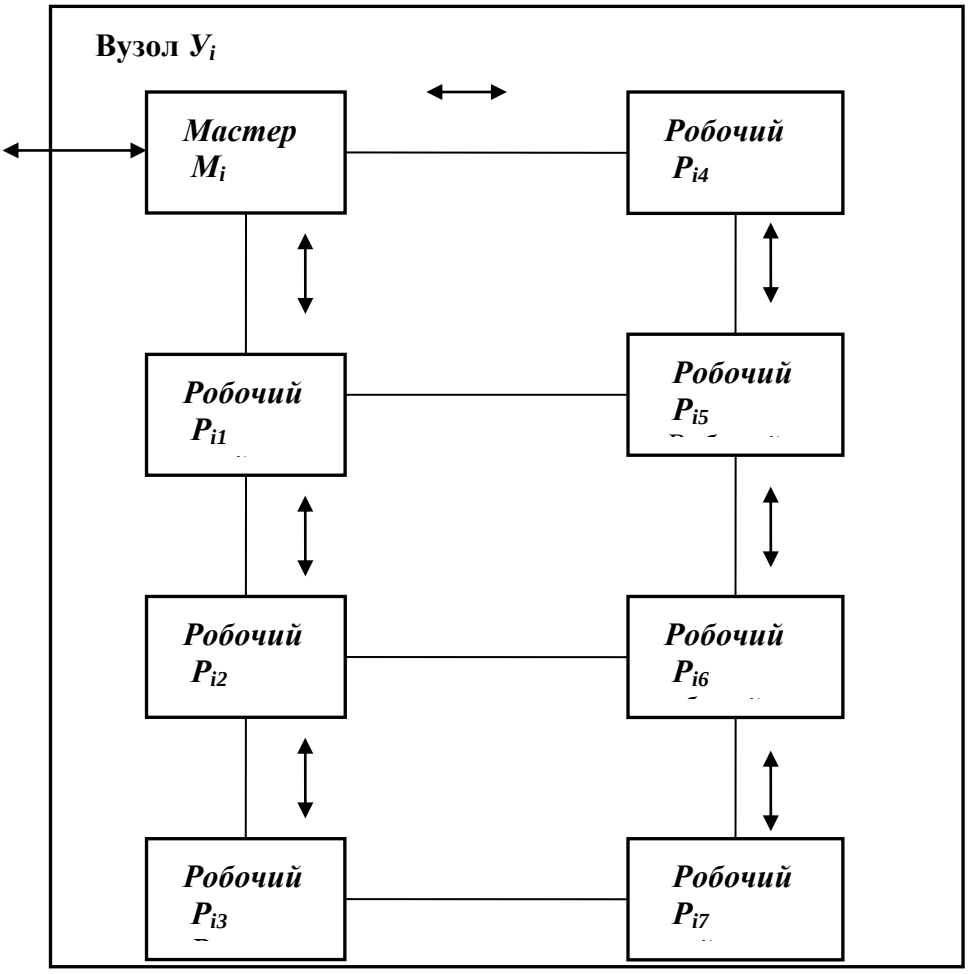


Рисунок 2.5 – Схема 1 взаємодії об’єктів в моделі КСБА2

У вузлі з топологією решітка поведінку об'єкта майстер описується наступним чином:

M_i = (M_i .*передати*.$P_{i1} \rightarrow M_i$.*передати*.$P_{i4} \rightarrow M_i$.*обчислення* $\rightarrow$

M_i .*прийняти*.$P_{i4} \rightarrow M_i$.*прийняти*.P_{i1})

У вузлі з архітектурою типу «решітка» поведінку об'єкта робочий описується наступним чином:

P_{i1} = (P_{i1}.*прийняти*.$M_i \rightarrow P_{i1}$.*передати*.$P_{i2} \rightarrow P_{i1}$.*обчислення* $\rightarrow$

P_{i1}.*прийняти*.$P_{i2} \rightarrow P_{i1}$.*передати*.$M_i$)

P_{i2} = (P_{i2} .*прийняти*.$P_{i1} \rightarrow P_{i2}$.*передати*.$P_{i3} \rightarrow P_{i2}$.*обчислення* $\rightarrow$

P_{i2}.*прийняти*.$P_{i3} \rightarrow P_{i2}$.*передати*.$P_{i1}$)

P_{i3} = (P_{i3} .*прийняти*.$P_{i2} \rightarrow P_{i3}$.*обчислення*, $\rightarrow P_{i3}$.*передать*.P_{i2})

P_{i4} = (P_{i4} .*прийняти*.$M_i \rightarrow P_{i4}$.*передати*.$P_{i5} \rightarrow P_{i4}$.*обчислення* $\rightarrow$

P_{i4} .*прийняти*.$P_{i5} \rightarrow P_{i4}$.*передати*.$M_i$)

P_{i5} = (P_{i5}.*прийняти*.$P_{i4} \rightarrow P_{i5}$.*передать*.$P_{i6} \rightarrow P_{i5}$.*обчислення* $\rightarrow$

P_{i5}.*прийняти*.$P_{i6} \rightarrow P_{i5}$.*передати*.$P_{i4}$)

P_{i6} = (P_{i6}.*прийняти*.$P_{i5} \rightarrow P_{i6}$.*передати*.$P_{i7} \rightarrow P_{i6}$.*обчислення* $\rightarrow$

P_{i6}.*приняти*.$P_{i7} \rightarrow P_{i6}$.*передати*.$P_{i5}$)

P_{i7} = (P_{i7} .*прийняти*.$P_{i6} \rightarrow P_{i7}$.*обчислення* $\rightarrow P_{i7}$.*передати*.P_{i6})

Особливістю моделі КСБА2 з архітектурою типу «решітка» є те, що взаємодія об'єктів можна описати кількома способами. Це пов'язано з тим, що архітектура типу «решітка» характеризується збільшенням числа зв'язків, що дозволяє реалізувати декілька варіантів передачі даних між об'єктами робітник. Це є основою забезпечення надійності КСБА при виході з ладу ядер або зв'язків. На рисунку 2.6 приведений інший варіант поведінки об'єктів.

P_{i1} = (P_{i1} .*прийняти*.$M_i \rightarrow P_{i1}$.*передати*.$P_{i2} \rightarrow P_{i1}$.*передати*.$P_{i5} \rightarrow$

P_{i1}.*обчислення* $\rightarrow P_{i1}$.*прийняти*.$P_{i5} \rightarrow P_{i1}$.*прийняти*.$P_{i2} \rightarrow P_{i1}$.*передати*.$M_i$)

P_{i2} = (P_{i2} .*прийняти*.$P_{i1} \rightarrow P_{i2}$.*передати*.$P_{i3} \rightarrow P_{i2}$.*передати*.$P_{i6} \rightarrow$

P_{i2}.*обчислення* $\rightarrow P_{i2}$.*прийняти*.$P_{i6} \rightarrow P_{i2}$.*прийняти*.$P_{i3} \rightarrow P_{i2}$.*передати*.$P_{i1}$)

P_{i3} = (P_{i3} .*прийняти*.$P_{i2} \rightarrow P_{i3}$.*передати*.$P_{i7} \rightarrow P_{i3}$.*обчислення* $\rightarrow$

P_{i3}.*приняти*.$P_{i7} \rightarrow P_{i3}$.*передати*.$P_{i2}$)

P_{i4} = (P_{i4} .*прийняти*.$M_i \rightarrow P_{i4}$.*обчислення* $\rightarrow P_{i4}$.*передати*.M_i)

P_{i5} = (P_{i5} .*прийняти*.$P_{i1} \rightarrow P_{i5}$.*обчислення* $\rightarrow P_{i5}$.*передати*.P_{i1})

P_{i6} = (P_{i6} .*прийняти*.$P_{i2} \rightarrow P_{i6}$.*обчислення* $\rightarrow P_{i6}$.*передати*.P_{i2})

P_{i7} = (P_{i7} .*прийнятиь*.$P_{i3} \rightarrow P_{i7}$.*обчислення к* $\rightarrow P_{i7}$.*передати*.P_{i3})

У вузлі з архітектурою гіперкуб поведінка об'єкта *Хост* описується наступним чином:

M_i = (M_i.*передати*.$X_i P_{i1} \rightarrow M_i$.*передати*.$P_{i3} \rightarrow$

M_i .*передати*.$X_i P_{i7} \rightarrow M_i$.*обчислення* $\rightarrow M_i$.*прийняти*.$P_{i7} \rightarrow$

M_i .*прийняти*.$P_{i3} \rightarrow M_i$.*прийняти*.P_{i1})

У вузлі з архітектурою гіперкуб поведінка об'єкта робочий описується наступним чином:

P_{i1} = (P_{i1}.*прийняти*.$M_i \rightarrow P_{i1}$.*передати*.$P_{i2} \rightarrow P_{i1}$.*передати*.$P_{i3} \rightarrow$

P_{i1}.*счет*$_1 \rightarrow P_{i1}$.*прийнять*.$P_{i3} \rightarrow P_{i1}$.*прийняти*.$P_{i2} \rightarrow P_{i1}$.*передати*.$M_i$)

P_{i2} = (P_{i2} .*прийняти*.P_{i1} → P_{i2}.*передати*.$P_{i2}P_{i5}$ →

P_{i2}.*обчислення* .P_{i2} → P_{i2}.*прийняти*.P_{i5} → P_{i2}.*передати*.P_{i1})

P_{i3} = (P_{i3} .*прийняти*.M_i → P_{i3} .*передати*.P_{i6} →

P_{i3}.*счет* → P_{i3}.*прийняти*.P_{i6} → P_{i3}.*передати*.M_i)

P_{i4} = (P_{i4}.*прийняти*.P_{i1} → P_{i4}.*обчислення* → P_{i4}.*передати*.P_{i1})

P_{i5} = (P_{i5}.*прийняти*.P_{i2} → P_{i5}.*обчислення* → P_{i5}.*передати*.P_{i2})

P_{i6} = (P_{i6} .*прийняти*.P_{i3} → P_{i6}.*обчислення* → P_{i6}.*передати*.P_{i3})

P_{i7} = (P_{i7} .*прийняти*.M_i → P_{i7}.*обчислення* → P_{i7}.*передати*.M_i)

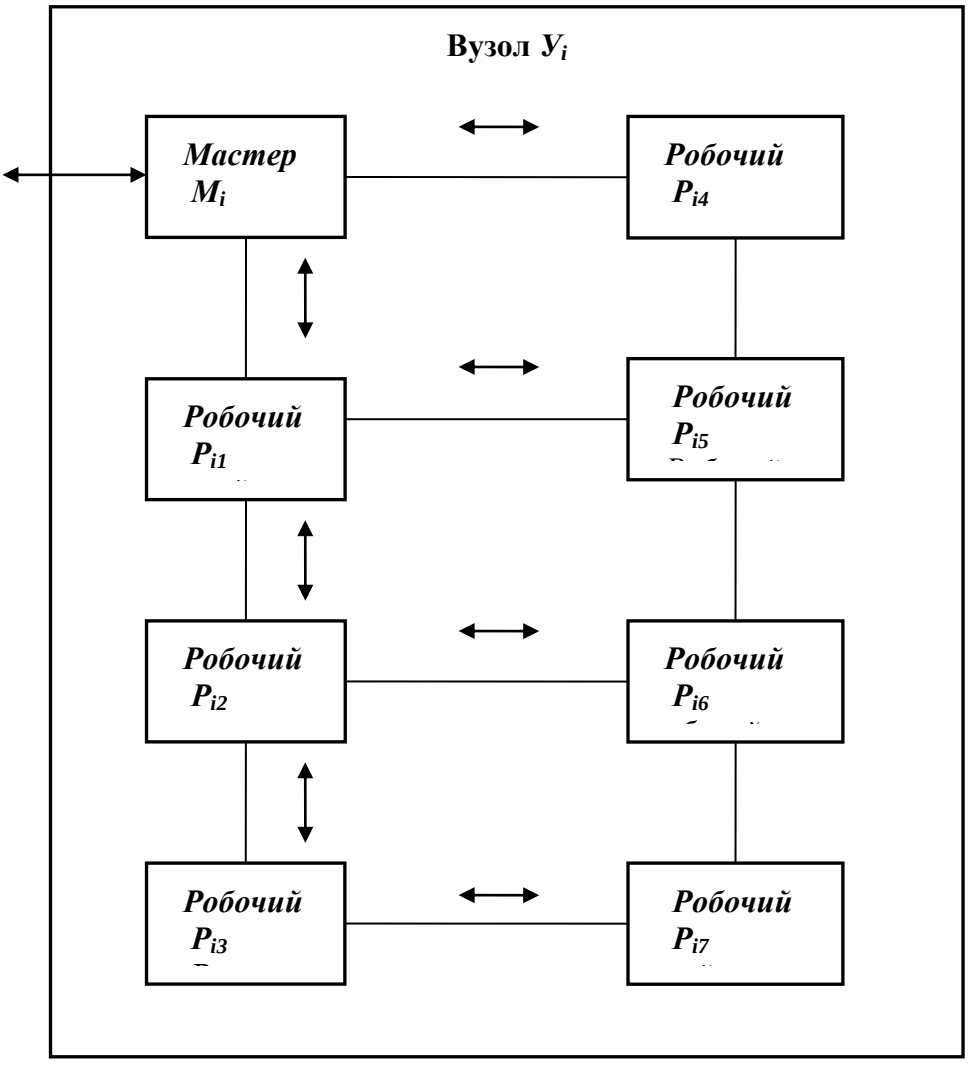


Рисунок 2.6 – Схема 2 взаємодії об’єктів в моделі КСБА2

Таким чином, кошти *CSP* теорії дозволяють сформувати алфавіти об'єктів і описати їх поведінку об'єктів для різних можливих видів зв'язку об'єктів у вузлі КСБА.

Реалізація моделі передачі повідомлень в КСБА2 виконується за допомогою об'єктів комунікації, розглянутих в 2.2. Відмінність полягає в тому, що для передачі даних між процесами всередині не використовуються мережеві технології і не потрібна інформація, пов'язана з визначенням місця розташування вузла. Для передачі даних досить знати ім’я процесу, так як всі процеси у вузлі виконуються під управлінням однієї операційної системи.

В описі поведінки у вузлі об'єктів майстер і робочий їх взаємодії задані наступними подіями:

Мастер: M_i .*передати*.$P_{ік}$ M_i .*прийняти*.$P_{ік}$

Робочий к: $P_{ік}$.*прийняти*.M_i , $P_{ік}$.*передати*.M_i ,

$P_{ік}$.*передати*.P_{im} , $P_{ік}$. *прийняти*.P_{im}

Модель, заснована на загальних змінних. Модель, заснована на загальних змінних, пов'язана з необхідністю вирішення двох задач: задачі взаємного виключення і задачі синхронізації [12]. Задача взаємного виключення припускає організацію взаємовиключного доступу до загальних змінних (загальних ресурсів), задача синхронізації – узгодження поведінки об'єктів.

У *CSP* теорії взаємодія процесів через загальні змінні не розглядається, так використовується модель, заснована на передачі повідомлень і її реалізація через канали.

Класифікація засобів взаємодії процесів. При класифікації засобів взаємодії, заснованих на загальних змінних, розглядатимемо їх по відношенню до контролю процесу або загального ресурсу. В цьому випадку такі засоби можна розділити на два класи. До першого класу відносяться засоби, побудовані на контролі процесів (див. рисунок 1.6). Їх реалізація заснована на виділенні в процесах критичних ділянок (КД), в яких здійснюється звернення до загальних ресурсів. В цьому випадку контроль за використанням загальних ресурсів зводиться до контролю безпосередньо критичних ділянок. Для цього навколо КД розставляються примітиви ВХОДКУ і ВИХОДКУ, які контролюють можливість входу в КД (ВХОДКУ) і фіксують вихід їх КД (ВИХОДКУ). При цьому сам загальний ресурс не контролюється. Алгоритми примітивів ВХОДКУ і ВИХОДКУ:

ВХОДКУ: перевірити, чи не знаходиться інший процес в своїй КД. Якщо

знаходиться, то блокувати даний процес. Інакше - встановити заборону для всіх процесів по входу в КД і процес може входити в КД.

ВИХОДКУ: зняти загальну заборону на вхід кожного процесу в КД.

При такому підході необхідно на етапі формування алгоритмів процесів виділяти в ньому критичні ділянки. Прикладами механізмів, що реалізовують такий підхід є семафори, мютекси, події. Розвитком підходу, заснованого на контролі процесів, є реалізація примітивів ВХОДКУ і ВИХОДКУу вигляді єдиної конструкції. Це дозволить уникнути помилок, пов'язаних з некоректним використанням примітивів ВХОДКУ і ВИХОДКУ. Такий підхід реалізований в механізмі критичні секції (Win32), синхронізовані блоки (Java), замки і монітори (C#).

Другий клас засобів взаємодії процесів через загальні змінні заснований на контролі загальних ресурсів (див. рисунок 1.7). Загальний ресурс (загальна змінна) інкапсулюється в програмний модуль (монітор) і доступ до нього можливий тільки через процедури (методи) цього модуля [34]. При цьому процедури монітора забезпечують синхронізований режим виконання, при якому може виконуватися тільки одна моніторна процедура. Механізм моніторів реалізований в мові Ада у вигляді захищених модулів [35]. Побудова моніторів на основі синхронізованих (synchronized) методів можливо в мові Java [15].

Об'єкти синхронізації. Введемо поняття об'єкту синхронізації (ОС), який використовуватиметься процесами при взаємодії через загальну пам'ять. На основі аналізу існуючих механізмів синхронізації, виділимо три види об'єктів синхронізації.

Перший вид об'єктів синхронізації (*ОС.А*) пов'язаний з організацією неподільних операцій над змінними.

Другий вид об'єктів синхронізації (включає два типи об'єктів синхронізації *ОС.S* і *ОС.К*) грунтується на контролі процесів, які звертаються до загальних змінними. Третій вид об'єктів синхронізації ОС.*М* грунтується на контролі загальних ресурсів.

Алфавіт об'єкту синхронізації *ОС.А* включає дії, пов'язані з виконанням певних дій над загальними змінними:

$$\alpha A = \{A.опис(x)\}$$

де подія *A.опис* припускає опис змінної (*x*) як неподільною змінною. При цьому будь-яке звернення до цієї змінної, як для читання, так і для зміни виконується процесами у взаємовиключному режимі.

Алфавіт об'єкту синхронізації типу *S* визначає набір подій, які дозволяють контроль процесу при використанні ним загальних змінних:

αS = {S.встановити_сгнл, S.провірити, S.встановити_несгнл}

де

• S.*встановити_сгнл* - встановлення *S* в сигнальний стан.

• *S.провірити* - перевірка стану *S*; блокування процесу при зайнятому (несигнальному) стані об'єкта *S*; встановлення *S* в сигнальний стан при сигнальному стані;

• *S.встановити_несгнл* - встановлення *S* в несигнальний стан.

Алфавіт об'єкта синхронізації типу *K* визначає набір подій, які також дозволяють контроль процесу при використанні ним загальних змінних:

αK = {K.захватити (T)}

де *K.захватити* стан, який визначається наступними діями:

– перевірка стану *K*;

– блокування процесу при зайнятому стані об'єкта *K*, захоплення *K* при вільному стані,

– виконання операції *T*, яка визначається подією в батьківському процесі (*T. обчислення*),

– перемикання стану об'єкта *K* в незайнятий стан.

Алфавіт об'єкта синхронізації типу *M* визначає набір подій, які також дозволяють контроль загального ресурсу:

αM = {M.запис, M.читання, M.обчислення, M.очікування, M.сигнал, M.несигнал}

де події:

- *M.запис* - запис даних в ОС,
- *M.читання* - читання (копіювання) даних з ОС,
- *M.обчислення* - обчислення безпосередньо в ОС,

- *М.очікування* - очікування сигналу в ОС,
- *М.несигнал* - встановлення несигнального стану в ОС.
- *М.сигнал* - встановлення сигнального стану в ОС.

Меню початкового стану *В* для *ОС.М* становлять події {*М.сигнал, М.несигнал*}.

Поведінку об'єктів синхронізації розглядатимемо як поведінку підлеглого процесу, визначеного в теорії Ч.Хоара. При цьому основним процесом виступає процес, який використовує *ОС* для взаємодії. Для основного (*Х*) і підлеглого (*К*) процесів має місце співвідношення $K // X = (K || X) \setminus \alpha K$. При цьому $\alpha K \subseteq \alpha X$ і $\alpha(K // X) = (\alpha K - \alpha X)$.

При взаємодії процесів *М* і *Р* через іменований *ОС а* діє співвідношення (*а: ОС // (М || Р*)). Підпроцес *а: ОС* розглядається як ресурс, який розділяється процесами *М* і *Р* і використовується для взаємодії. Важливою особливістю поведінки процесів *М* і *Р* є взаємовиключний доступ до *а*, що в теорії Ч.Хоара позначається як ($M ||| P$).

Розглянемо використання цих об'єктів синхронізації при організації взаємодії процесів, яке пов'язане з вирішенням задачі взаємного виключення і синхронізації.

Об’єкт *ОС.А* забезпечує тільки вирішення задачі взаємного виключення. Процес, який описує поведінку об’єкту *А*, виконує подію *А.опис*, а потім веде себе як *А*:

$$A = (A.опис(x) \rightarrow A)$$

Звернення процесів до змінною (*х*), описано] в *ОС.А* пов'язано з виконанням будь-яких дій над цією змінною, які виконуються у взаємовиключному режимі.

Об'єкт *ОС.S* забезпечує вирішення обох задач взаємодії процесів. Процес, який описує поведінку об'єкту *S*, пов'язаний з встановленням *S*, а потім - з виконанням дій, які пов'язані з початковою перевіркою і зміною стану *ОС.S*. Якщо *В* – множина подій, *х* – подія з множини *В*, *Р(х)* – процес з початковим станом *х*, то множина *В* є початковим меню процесу *Р* : ($x: B \rightarrow P(x)$).

Для об'єкту *ОС.S* має місце початкове меню:

$B = \{$ *dстановитьь_сгнл, dстановитb_несгнл* $\}$.

Поведінка об'єкту *S*:

S = ((*S.Встановити_сгнл* | *S.Встановити_несгнл*) $\rightarrow$

((*S.встановити_сгнл* | *S.провірити*) $\rightarrow S$)))

Якщо процеси *M* і *P* виконують події *X.обчислення* і *P.обчислення,* пов'язані із загальним ресурсом, то обидва процеси здійснюють ці обчислення в критичній ділянці, яка захищається *ОС.S* (*встановити_сгнл*) :

X: (*S.провірити* $\rightarrow$ *M.обчислення* $\rightarrow$ *S.встановити_сгнл*)

P: (*S.проверить* $\rightarrow$ *P.обчислення* $\rightarrow$ *S.встановити_сгнл*)

При синхронізації процесів, коли об'єкт *X* посилає сигнал об'єкту *P* про подію, використовується процес *ОС.S* (*встановити_несгнл*) взаємодія пов'язана з такими подіями в процесах *X* і *P*:

X: (*S.провірити*), *P:* (*S.встановити_сгнл*)

Об'єкт *ОС.К* призначений виключно для вирішення задачі взаємного виключення:

K = ((*К.захватити* (*T*)) $\longrightarrow$ *K*),

де за допомогою параметра *T* задаються дії, які процес виконує в критичній ділянці.

Якщо процеси *X* і *P* виконують події *X.обчислення* і *P.обчислення,* пов'язані із загальним ресурсом, то обидва процеси здійснюють ці обчислення в критичній ділянці, яка захищається *ОС.К:*

X: (. . . $\longrightarrow$ *К.захватити* (*X.обчислення*) $\longrightarrow$ . . .)

P: (. . . $\longrightarrow$ *К.захватити* (*P.обчислення*) $\longrightarrow$ . . .)

Відмінність об'єкту *ОС.К* від об'єкту *ОС.S* полягає в тому, що події *S.захватити* і *S.звільнити* об'єднані в одну дію *К.захвати(Т)*. Це дозволяє інкапсулювати дії із загальним ресурсом всередину *ОС.К* і уникнути помилок, які виникають при використанні *ОС.S,* коли вхід і вихід в критичну ділянку реалізований через окремі дії.

Об'єкт *ОС.М* забезпечує вирішення задач взаємного виключення і синхронізації. Поведінка об'єкту *M* при вирішенні задачі взаємного виключення:

M = ((*M.запис* | *M.читання* | *M. обчислення*) $\longrightarrow$ *M*)

Звернення процесів до загального ресурсу, що знаходиться в *ОС.М*

пов'язано з такими подіям в процесах *X* і *P*:

X: (*M.запис* $\longrightarrow \ldots \longrightarrow$ *M.читання* $\longrightarrow \ldots \longrightarrow$ *M.обчислення*) $\longrightarrow$

P: (*M.читання* $\longrightarrow \ldots \longrightarrow$ *M.запис* $\longrightarrow \ldots \longrightarrow$ *M.обчислення*) $\longrightarrow$

Синхронізація процесів через *ОС.М* пов'язана з такими діями в процесах *X* і *P*, якщо процес *X* чекає сигналу від процесу *P*:

X: (*M.сигнал*), *P:* (*M.чекати*)

При цьому поведінка об'єкту *ОС.М* описується як

M = ((*M.чекати* | *M.сигнал*) $\longrightarrow$ *M*)

Розглядаючи *ОС* як підлеглий процес, можна сформулювати правила щодо алфавітів і поведінки процесів, що взаємодіють через *ОС*, які визначають коректність взаємодії процесів.

Наступне правило визначає необхідну умову взаємодії двох процесів по відношенню до алфавітів процесів і алфавіту *ОС*.

“Якщо *P* і *R* - процеси, *E* - об'єкт синхронізації, то необхідною умовою взаємодії об'єктів *P* і *Q* через *ОС*. *E* є виконання співвідношення $\alpha E \subseteq (\alpha P \cup \alpha Q)$“.

Наступні правила визначають вимоги до поведінки процесів і об'єктів синхронізації.

Наступне правило визначає необхідну умову взаємодії двох процесів по відношенню до поведінки процесів і поведінки *ОС* типу *А*.

"Якщо *P* і *R* - процеси, *e* - об'єкт комунікації типу *ОС.А*, то необхідною умовою взаємодії *P* і *Q* через *ОС.А (e)*, є виконання наступних подій, що визначають поведінку цих процесів:

$$A(e) = ((A.опис(e)) \rightarrow (A) \quad (A(e) // (P(e) \| Q(e))).$$

Наступне правило визначає необхідну умову взаємодії двох процесів по відношенню до поведінки процесів і поведінки ОС типу S.

“Якщо *P* і *R* - процеси, *Z* - об'єкт комунікації типу *ОС.S(Z)*, то необхідною умовою взаємодії *P* і *Q* через *ОС.А(Z)* є виконання наступних подій, що визначають поведінку цих процесів:

а) для задачі синхронізації де *Q* чекає події в *P*

Z(*встановити_несгнл*)= ((*Z.провірити* | *Z.встановити_сгнл*) $\longrightarrow$ *Z*)

P = ((*P.подія* $\longrightarrow$ *Z.встановити_сгнл*)),

Q = ((*P.очікування_події* $\longrightarrow$ *Z.провірити*)) $\longrightarrow$

(Z // (P || Q)).

б) для задачі взаємного виключення

Z(*встановити_несгнл*)= ((*Z.провірити* | *Z.встановити_сгнл*) $\rightarrow$ Z),

P = (*Z.провірити* $\rightarrow$ *P.обчислення* $\rightarrow$ *Z.встановити_сгнл*)

Q = (*Z.провірити* $\rightarrow$ *Q.обчислення* $\rightarrow$ *Z.встановити_сгнл*)

(Z // (P || Q)).

Наступне правило визначає необхідну умову взаємодії двох процесів по відношенню до поведінки процесів і поведінки *ОС* типу *К*.

"Якщо *P* і *R* - процеси, *К* - об'єкт комунікації типу *ОС.К*, то необхідною умовою взаємодії *P* і *Q* через *ОС.К* є виконання наступних подій, що визначають поведінку цих процесів:

К = ((*К.захватити* ()) $\rightarrow$ *К*)

Q = (... $\rightarrow$ *К.захватити* (*Q.обчислення*) $\rightarrow$ ...)

P = (... *К.захватити* (*P.обчислення*) $\rightarrow$ ...)

(*К* // (*P* || *Q*)).

І наступне правило визначає необхідну умову взаємодії двох процесів по відношенню до поведінки процесів і поведінки *ОС* типу *К*.

“Якщо *P* і *R* - процеси, *M* – об’єкт комунікації типу *ОС.М*, то необхідною умовою взаємоддії *P* і *Q* через *ОС.М*, є виконання наступних подій, які визначають поведінку цих процесів:

а) для задачі синхронізації де *Q* очікує події в *P*

M(*несигнал*) = ((*M.чекати* | *M.сигнал*) $\longrightarrow$ *M*)

P = ((*P.подія* $\longrightarrow$ *M.сигнал*)) ,

Q = ((*M.чекати* $\longrightarrow$ *Q.обчислення*))

(M // (P || Q)).

б) для задачі взаємного виключення

M(*сигнал*) = ((*M.запис* | *M.читання* | *M. обчислення*) $\longrightarrow$ *M*)

X = (*M.обчислення* | *M.запис* | *M.обчислення*)

P = (*M.читання* | *M.запис* | *M.обчислення*)

(M // (P || Q)).

Взаємодію моделей першого і другого рівнів КСБА можна здійснити різними способами залежно від моделі програмування, яка використовуватиметься і вибору об'єкту комунікації.

Найбільш раціональним є рішення, засноване на тому, що об'єкти другого рівня розглядаються як вкладені в об'єкти першого рівня (сервер і клієнт). В цьому випадку об'єкт першого рівня, наприклад, клієнт після отримання даних від об'єкту сервер ініціює паралельне виконання об'єктів, що описують модель другого рівня в кожному вузлі.

На рисунку 2.7 представлена схема взаємодії моделей першого і другого рівнів. Після отримання *і*-м клієнтом даних від сервера він ініціює запуск об'єктів другого рівня. На рисунку 2.7 об'єктів другого рівня представлені об'єктом майстер (M_i), об'єктами робочий ($P_{ік}$) і об'єктом синхронізації *ОС.М.*

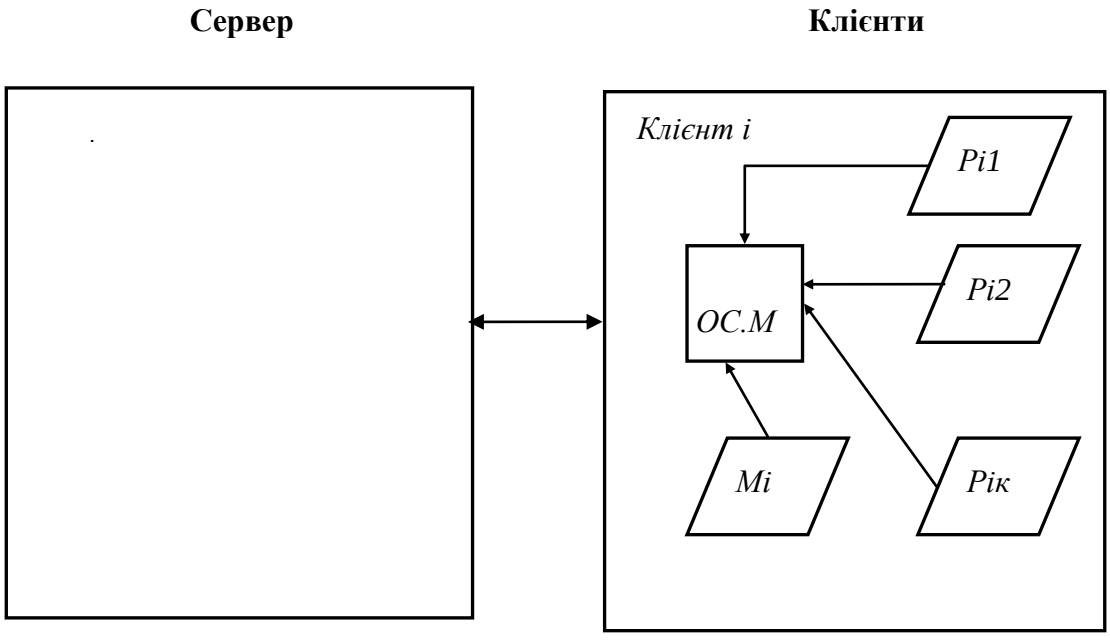


Рисунок 2.7 – Схема зв’язування об’єктів в моделях *КСБА1* і *КСБА2*

Взаємодія об'єктів другого рівня виконується через *ОС.М*, який також може бути використаний і для організації взаємодії з першим рівнем у вузлі. Як правило, взаємодія між рівнями зводиться до синхронізації по отриманню даних від сервера до вузла і синхронізації по завершенню обчислень у вузлі, після чого на першому рівні виконується передача результату серверу.

Отже, в даному розділі розроблено математичні моделі обчислень першого

та другого рівнів для комп’ютерних систем з багатоядерною архітектурою, які дозволяють описати дворівневу організацію обчислень в КСБА і виконати роль інтерфейсу між архітектурою системи та моделями програмування, що спрощує розробку програмних компонент для КСБА.

3 РЕАЛІЗАЦІЯ МОДЕЛЕЙ ОБЧИСЛЕНЬ

3.1 Реалізація моделі обчислень першого рівня в кластерних системах з багатоядерною архітектурою

Реалізація моделі першого рівня КСБА1 пов'язана з двома задачами: організацією процесів у вузлах і з організацією взаємодії вузлів.

Організація процесів. При організації розподілених обчислень в моделі КСБА1 виходитимемо з концепції, запропонованої в [36]. Розподілену систему розглядатимемо як набір вузлів, які з’єднанні мережею. При цьому вузли діляться на вузли обробки і вузли зберігання.

Розподілена програма розглядається як набір розділів. Розділи виконуються незалежно на вузлах розподіленої системи і можуть взаємодіяти між собою. Розділ формується з бібліотечних модулів, тобто модулів, що пройшли компіляцію. Основою побудови розділу є пакет (package). У розділі може бути об'єднано декілька пакетів.

Програмне забезпечення північного вузла пов'язане з основною процедурою, з якою починається виконання розподіленого застосування. Багатопоточність сервера грунтується на використанні спеціальних модулів мови – задач (task), які дозволяють організувати одночасне виконання модулів, тобто паралельну обробку. На рівні сервера задачі використовуються для організації взаємодії з клієнтами.

Важливою особливістю формування розподіленого застосування є вимога мінімальних перетворень при переході від звичайного додатку до розподіленого застосування.

Взаємодія розділів. Реалізація взаємодії вузлів в КСБА може бути реалізована за допомогою механізму сокетів і механізму віддалених методів [4].

Сокети - низькорівневий механізм, реалізований в різних комутаційних бібліотеках (наприклад, в Win32), а також вбудований в мови програмування (Java) [23].

У мові Ада реалізація взаємодії розділів грунтується на механізмі виклику

віддалених процедур (Remote Procedure Call - RPC механізмі). На рисунку 3.1 представлена схема взаємодії сервера і клієнтів на основі RPC механізму. Серверна частина побудована на використанні задач, кожна з яких відповідає за взаємодію зі своїм клієнтом. Взаємодія полягає у виклику віддаленої процедури *Обчислення()*, під час якого відбувається передача даних від сервера до клієнта, на стороні клієнта виконується тіло віддаленої процедури, пов'язане зі лічильною частиною розподіленого додатку і потім результат повертається від клієнта до сервера (маршалінг).

Виклик віддаленої процедури може бути здійснено прямим викликом як виклик процедури, описаної в пакеті з віддаленим інтерфейсом, так і непрямим викликом з використанням посилальних типів (remote access-to-subprogram type). Прямий виклик визначає статичне зв'язування того хто викликає і того кого викликає розділів, непрямий - динамічне зв'язування.

Обсяг даних визначається кількістю і типом параметрів процедури, напрямок - специфікаторами *in* і *out*. Розділ, що визначає поведінку клієнта, базується на пакеті *Дата*, який містить віддалену процедуру *Обчислення*. Нижче наведена специфікація пакета *Дата*, в якому описана процедура *Обчислення*, що не є віддаленою:

```
package Дата is
procedure Счет(МА,МВ:in Матриця; МС: out Матриця);
end Дата;
```

Для опису віддаленої процедури використовуються спеціальні прагми категорії. Прагма `Remote_Call_Procedure_Call([ім'я бібліотечного модуля)]` використовується для створення віддаленого інтерфейсу пакету. Процедура, описана в пакеті, до якого застосована прагма `Remote_Call_Interface`, є віддаленою і допускає віддалений виклик з іншого розділу.

```
package Дата is
pragma Remote_Procedure_Call;
procedure Рахунок(Ма,мв:in Матриця; МС: out Матриця);
end Дата;
```

Сервер Клієнти

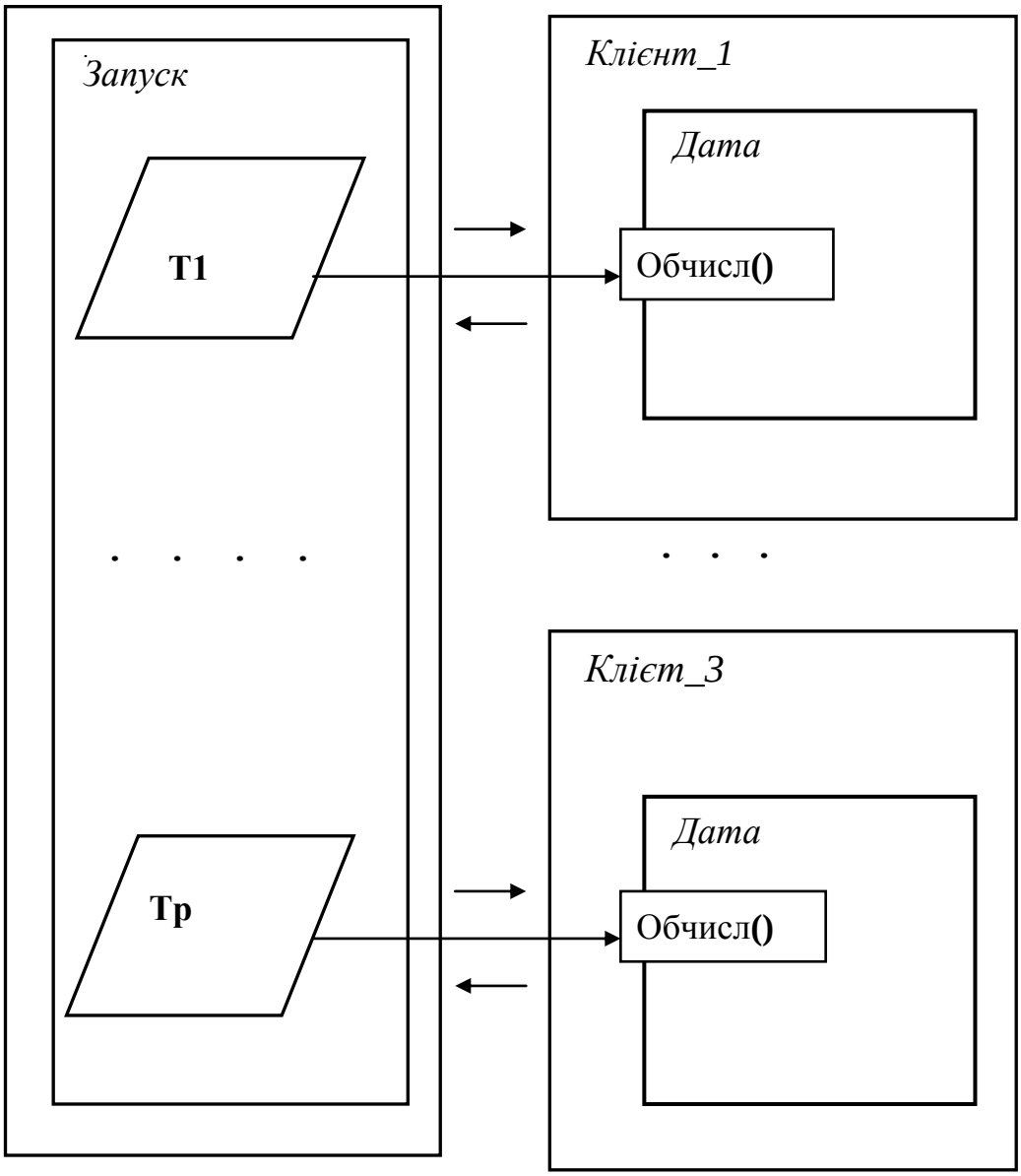


Рисунок 3.1 – Взаємодія в КСБА1 на основе RPC

Важливою складовою виду маршалінгу розділів є синхронна і асинхронна взаємодія. При синхронній взаємодії розділ, що викликає при виклику віддаленої процедури блокується і чекає отримання результату виконання віддаленого методу. При асинхронному виклику віддаленої процедури розділ, що викликає продовжує своє виконання відразу після завершення передачі параметрів в розділ, що викликається. Тобто без очікування формування і повернення результату виконання віддаленої процедури. Опис асинхронної взаємодії через віддалену процедуру виконується через прагму `Asynchronous(ім'я процедури)`. Для віддалених методів, що допускають асинхронний віддалений виклик, діє

обмеження, пов'язане з неможливістю використання в специфікації віддаленої процедури параметрів з модифікатором напряму `out`. Для процедури `Счет()`, описаною в пакеті `Дата`, асинхронний виклик неможливий. Нижче приведений приклад віддаленого інтерфейсу, що містить процедуру, що допускає віддалений асинхронний виклик.

```
package Кнл is
ragma Remote_Procedure_Call;
procedure Прийом(МХ : in Матриця; С : in Вектор);
pragma Asynchronous(Прийом);
end Кнл;
```

Таким чином, реалізація об'єкту комунікації здійснюється через віддалені процедури. Специфікація *ОК* визначається в цьому випадку специфікацією віддаленої процедури, де задаються об'єм і напрям передачі даних між сервером і клієнтом. Синхронна і асинхронні форми взаємодії задаються через спеціальні прагми.

Об'єкт комунікації в цьому випадку має наступні параметри (таблиця 3.1):

Таблиця 3.1 – Опис об'єктів комунікації

Об'єкт комунікації (ОК)			
Обчислення()	ОК.В		Синхронний - С
	ОК.С		
	ОК.С.Д		ОК.С.Н
	МА	Матриця	in
	МВ	Матриця	in
	МС	Матриця	out
Об'єкт комунікації (ОК)			
Прийом()	ОК.В		Асинхронній - А
	ОК.С		
	ОК.С.Д		ОК.С.Н
	МХ	Матриця	in
	С	Вектор	in

Буферизована форма взаємодії розділів в стандарті мови Ада не розглядається. Тобто *ОК.В = НБ.* Можлива її реалізація через додаткову задачу, яка виконуватиме роль буфера.

Стандарт мови Ада включає підсистему взаємодії розділів PCS (Partition Communication Subsystem), яка здійснює передачу і прийом даних клієнта, що

визиває і сервера через потік типу System.RPC.Para-ms_Stream_Type:

```
type Params_Stream_Type
(Initial_Size : Ada.Streams.Stream_Element_Count)
is new Ada.Streams.Root_Stream_Type with private;
```

Цей тип є своєрідним контейнером передачі даних між розділами розподіленої програми. Кореневим є тип Root_Stream_Type, який визначає базовий тип потоку і дві операції: Write (запис в потік) і Read (зчитування з потоку) об'єктів типу Stream_Element_Array, які є масивом байтів.

Читання і запис потоків здійснюється за допомогою використання чотирьох атрибутів: Write - записує елемент в потік, допустимо тільки для обмежених (constrained) типів; Read - читає обмежений (constrained) елемент з потоку; Output - те ж саме, що і Write, але, при необхідності, додатково записує в потік межі і дискримінанти; Input - те ж саме, що і Read, але додатково читає з потоку межі і дискримінанти.

Формування розділів, що визначають серверні і клієнтські об'єкти, здійснюється за допомогою файлів конфігурації (configuration files), в яких визначається склад розділу, місце розташування на обраному вузлі із зазначенням параметрів, необхідних для виконання розподіленої програми: IP адрес серверного і клієнтських об'єктів, виду запуску програми , зв'язки з іншими модулями та ін.

Нижче приведений приклад файлу конфігурації для формування серверного і трьох клієнтських об'єктів для схеми на рисунку 3.1:

```
configuration Файл_конфіг is
 -- формування серверного розділу
 Сервер: Partition := ();
 procedure Master_Procedure is in Сервер;

-- формування трьох клієнтських розділів
 Клієнт_1, Клієнт_2, Клієнт_3 : Partition;
 -- розміщення розділів по вузлах
 for Клієнт_1'host use "Вузол1";
 for Клієнт_2'host use "Вузол2";
 for Клієнт_3'host use "Вузол16";

 -- вказівка розташування виконуваних файлів
 for Клієнт_1'directory use "/prg1/bin";
 for Клієнт_2'directory use "/prg2/bin";
 for Клієнт_3'directory use "/prg3/bin";

 -- вид запуску застосування
```

```
 pragma Starter (Method => Ada);

 -- опис каналів для взаємодії розділів
 Канал_1 : Channel := (Сервер, Клієнт_1);
 Канал_2 : Channel := (Сервер, Клієнт_2);
 Канал_3 : Channel := (Сервер, Клієнт_3);

 -- опис фільтрів
 for Канал_1'filter use "ZIP";
 for Канал_2'filter use "ZIP";
 for Канал_3'filter use "ZIP14";

begin

 -- формування вмісту розділів
 Клієнт_1 := (Дата);
 Клієнт_2 := (Дата);
 Клієнт_3 := (Дата);

end MyConfig;
```

3.2 Реалізація моделі обчислень другого рівня в кластерних системах з багатоядерною архітектурою

Реалізація моделі КСБА2 пов'язана з реалізацією процесів, що визначають обчислення у вузлах, і їх взаємодією за допомогою об'єктів комунікації або синхронізації.

Реалізація процесів в Аді здійснюється за допомогою спеціальних модулів – задач (task). Мова забезпечує розвинені засоби для роботи з процесами, які дозволяють:

- описати задачу або групу задач;
- встановити пріоритет задачі;
- розмістити задачу на вибраному процесорі;
- запустити задачу на виконання;
- завершити виконання задачі;
- організувати взаємодію задач.

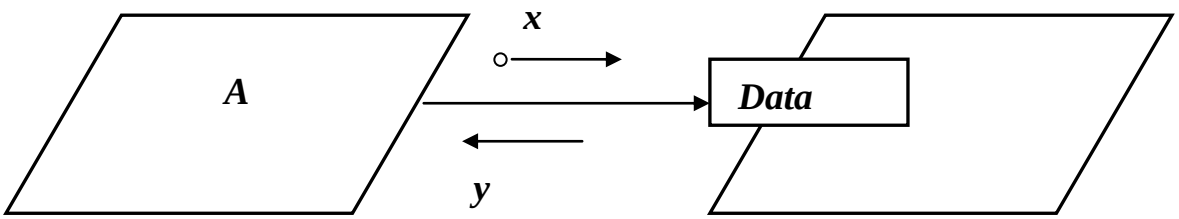
Взаємодія процесів. Організація взаємодії процесів пов'язана з реалізацією двох моделей: моделі, заснованої на загальних змінних і моделі, заснованої на передачі повідомлень.

Реалізація об'єктів комунікації. Основним інструментом реалізації *ОК* в мові Ада виступає механізм рандеву [27]. Механізм заснований на понятті входу

задачі, який визначає інтерфейс задачі для взаємодії з іншими задачами. На рисунку 3.2 представлена схема взаємодії двох задач з використанням механізму рандеву.

Основа взаємодії задач *A* і *B* – вхід `Data`, описаний в задачі *B*.

Нижче приведена програма, що реалізовує взаємодію задач *A* і *B* через вхід `Data`.



Рисуно 3.2 – Схема взаємодії задач в механізмі рандеву

```
procedure Prg is
 task A;
 task body A is
 x: integer := 100;
 y: integer;
 begin
 . . .
 B.Data(x, y);
 . . .
 end Data;
 task B is
 entry Data(t1: in integer; t2 : out integer);
 end B;
 task body B is

 x2: integer;
 y2: integer := 55;
 begin
 . . .
 accept Data(t1: in integer; t2 : out integer) do
 x2:= t1;
 t2:= t2;
 end Data;
 . . .
 end Data;
begin
 null;
```

```
end PRG;
```

Оператора входу можна використовувати для реалізації об'єкту комунікації *ОК*(*Д, Н, В, Б*). На прикладі взаємодії задач *A* і *B* через вхід `Data` розглянемо формування параметрів *ОК*:

- *ОК.Д* задається кількістю параметрів і їх типів, описаних в специфікації входу; для входу `Data` *ОК.Д* визначає передачу двох цілих чисел;

- *ОК.Н* задається модифікаторами *in* і *out*, які визначають напрям передачі інформації, описаній параметром *ОК.Д;* для входу `Data` *ОК.Н* визначає прийом і передачу двох чисел;

- *ОК.В* за замовчуванням задає синхронізовану взаємодію задач через вхід; асинхронна взаємодія додатково описується за допомогою оператора `select`;

- *ОК.Б* не припускає використання буферизованої взаємодії, яку можна реалізувати через додаткову задачу, яка виконуватиме роль буфера.

Таким чином, механізм рандеву дозволяє повною мірою реалізувати об'єкт комунікації, запропонований в розділі 2.2.

Комутаційна бібліотека MPI є найбільш поширеним інструментом для організації взаємодії процесів в кластерних системах [19]. Основою реалізації взаємодії процесів (задач) служать процедури `Send/Receive`, що дозволяють передавати і приймати дані. Існує декілька видів операцій `Send/Receive`, що забезпечують різноманітні форми передачі даних. У таблиці 3.2 представлений набір операцій `Send/Receive`. MPI реалізує попарний, колективний, синхронізований, буферизований види взаємодії. При реалізації об'єкту комунікації *ОК*(*Д, Н, В, Б*) для передачі повідомлення функція `Send` дозволяє встановити наступні параметри `MpI_[I] [R,S,B] Send : [I]` – неблокований режим; [R] – по готовності; `[S]` – синхронізований; `[B]` – буферизований. Стандартний обмін даними не вимагає префіксів. Важливою особливістю MPI є наявність групових операцій, яка реалізується за допомогою механізму комунікатора, який дозволяє об'єднати процеси в групу.

Реалізація об'єктів синхронізації. Об'єкти синхронізації, розглянуті в 2.3, призначені для організації взаємодії процесів при використанні моделі, заснованої на моделі загальних змінних. При цьому необхідне вирішення задач взаємного

виключення і синхронізації процесів.

Таблиця 3.2 – Функції бібліотеки MPI

Функція	Операція
`MPI_Send()`	Переслати повідомлення. Блокуватися, поки воно не буде отримано
`MPI_Recv()`	Отримати повідомлення. Блокуватися, поки воно не буде послано.
`MPI_ISend()`	Переслати повідомлення без блокування
`MPI_IRecv()`	Отримати повідомлення без блокування
`MPI_Probe()`	Встановити, чи отримано повідомлення (блокуватися до отримання повідомлення)
`MPI_IProbe()`	Встановити, чи отримано повідомлення (без блокування)
`MPI_Bcast()`	Послати повідомлення всіх процесів в групі
`MPI_Reduce()`	Отримати дані від всіх процесів в групі
`MPI_Gather()`	Зібрати дані від всіх процесів
`MPI_Scatter()`	Послати дані всім процесам у групі. Дані – одного розміру
`MPI_Scatterv()`	Послати дані всім процесам у групі. Дані можуть бути різного розміру
`MPI_Alltoall()`	Послати дані всім процесам в групі від всіх процесів групи. Дані одного розміру
`MPI_Alltoallv()`	Послати дані всім процесам в групі від всіх процесів групи. Дані можуть бути різного розміру

Об'єкт *ОС.А.* Об'єкт синхронізації типу *Ос.А* реалізується в мові Ада за допомогою атомік змінних, які описуються через прагми `Atomic` і `Volatile`. Два види прагм `Atomic` і `Atomic_Component` дозволяють забезпечити

синхронізований (що розділяється) доступ змінних по читанню і запису. Тобто операції над атомік змінними виконуються послідовно, сама операція не може бути перервана:

```
pragma Atomic(Проста_змінна)
pragma Atomic_Component(Масив_змінна).
```

Ще два види прагм `Volatile` і `Volatile_Component` забезпечують синхронізований доступ до змінних по читанню і запису, при цьому операції над змінними виконуються безпосередньо в пам'яті.

```
pragma Volatile(Проста_змінна)
pragma Volatile_Component(Масив_змінна).
```

Мови Java, C#, C++ також використовують об'єкти синхронізації типу *ОС.А*, реалізовані за допомогою модифікаторів `volatile` [15, 37, 38].

Об'єкт *ОС.S*. Об'єкт синхронізації типу *ОС.S* реалізується за допомогою механізмів семафорів, мютексів, подій. Це найбільш використовуваний вид об'єктів комунікації. Реалізовані в бібліотеці Win32, мові C# [38].

У мові Ада двійковий семафор пов'язаний з пакетом `Ada.Synchronous_Task_Control`. Пакет реалізує логічний тип для створення семафорного об'єкту (`Suspension_Object`), і операції, які необхідні при роботові з *ОС.К:*

S.встановити_сгнл =>

```
procedure Set_True(S: inout Suspension_Object)
```

S.провірити =>

```
procedure Suspend_Until_True(S: inout Suspension_Object)
```

S.встановити_несгнл =>

```
procedure Set_False(S: inout Suspension_Object).
```

Недолік семафорів, які використовуються в мові – орієнтація на взаємодію тільки двох задач. Це утруднює вирішення задачі взаємного виключення, якщо загальний ресурс використовується більше, ніж двома процесами, а також синхронізацію одного процесу з декількома. На рисунку 3.3 представлені види синхронізації процесів, що найчастіше виникають при взаємодії процесів. Це

взаємодія двох процесів, коли один процес чекає сигналу від іншого процесу про подію, яка має місце в цьому процесі (а)); коли один процес посилає сигнал декільком процесам (б)); коли один процес чекає сигнали від декількох процесів (в)). Тут введені точки синхронізації двох видів: `W` - точка очікування сигналу і `S` - точка передачі сигналу.

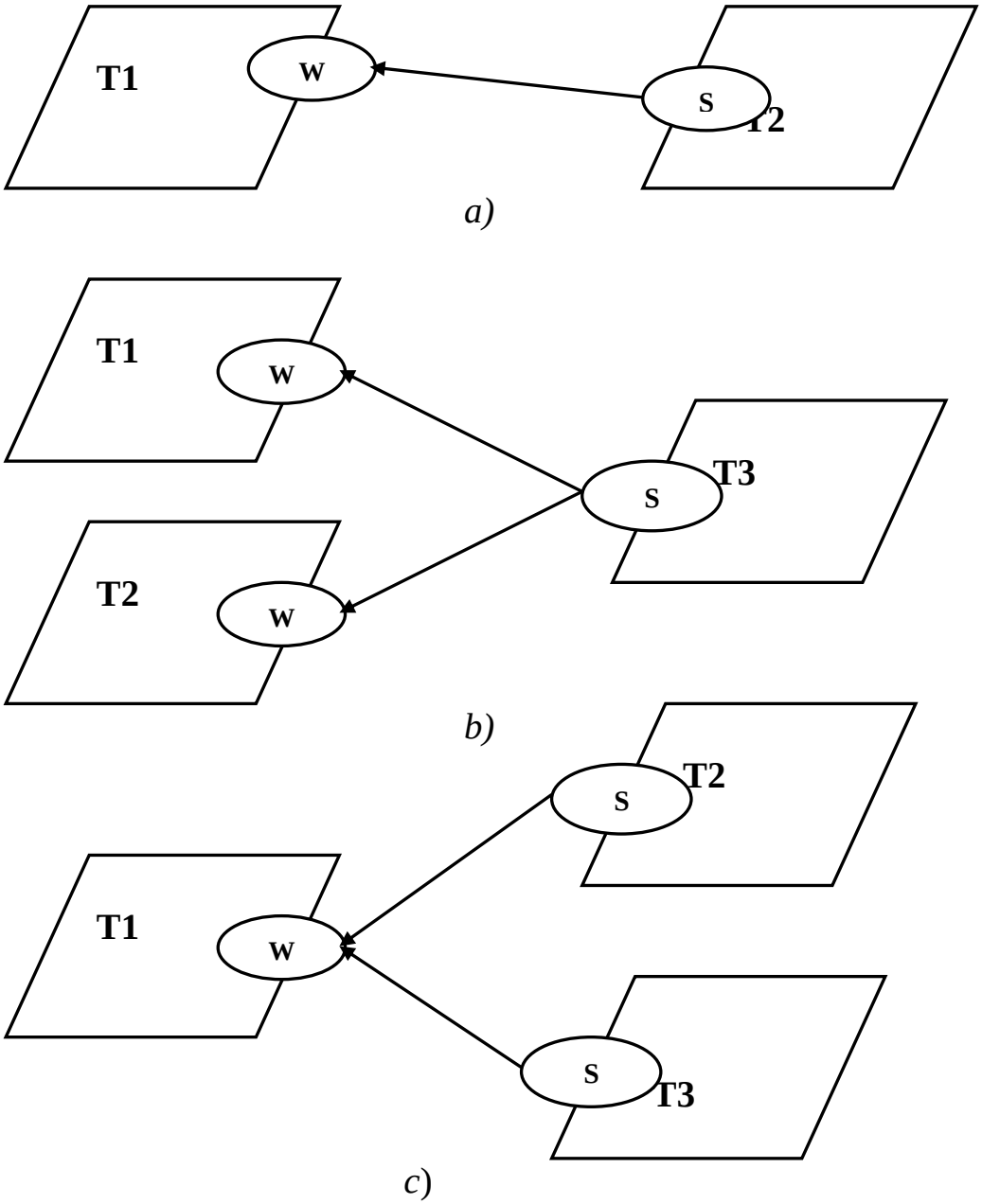


Рисунок 3.3 – Види синхронізації процесів

Множинні семафори, які використовуються в Win32 і C# позбавлені цих недоліків.

Для Win32 при використанні семафорів:

S.встановити_сгнл => `CreateSemaphore()`,

S.встановити_несгнл => `CreateSemaphore()`.

S.провірити => `WaitForSingleObject()`,

S.встановити_сгнл => `ReleaseSemaphore()`.

Функція `CreateSemaphore()` використовується при створенні об'єкту семафора на основі змінної типу `HANDLE`. Вона дозволяє вибір значення семафора з початкового меню *ОС.S*. Для *S.встановити_сгнл* використовуються наступні значення параметрів цієї функції:

S.встановити_сгнл => `CreateSemaphore(NULL, 1,1, NULL)`

S.встановити_несгнл => `CreateSemaphore(NULL, 0,1, NULL)`.

Для *S.провірити* і *S.встановити_сгнл* використовуються наступні значення параметрів цих функцій

S.провірити => `WaitForSingleObject (S, Time)`

S.встановити_сгнл => `ReleaseSemaphore(S, 1, NULL)`.

Для Win32 при використанні мютексів:

S.встановити_сгнл => `function CreateMutex()`,

S.встановити_несгнл => `function CreateMutex()`.

S.провірити => `function WaitForSingleObject()`,

S.встановитиь_несгнл => `function ReleaseMutex()`.

Функція `CreateMutex()` використовується при створенні об'єкту мютекс на основі змінної типу `HANDLE`. Вона дозволяє вибір значення мютекса з початкового меню. Для *S.встановити_сгнл* використовуються наступні значення параметрів `CreateMutex()`, для *S.встановити_несгнл* – `CreateMutex()`:

S.встановити_сгнл => `CreateMutex(NULL, 1, NULL)`

S.встановити_несгнл => `CreateMutex(NULL, 0, NULL)`.

Для *S.провірити* і *S.встановити_сгнл* використовуються наступні значення параметрів цих функцій

S.провірити => `WaitForSingleObject (S, Time)`

S.встановити_сгнл => `ReleaseSemaphore(S )`.

Об'єкт *ОС.К*. Об'єкти синхронізації типу *ОС.К* не реалізовані в мові Ада. Представлені в бібліотеці Win32, в мовах Java і C#.

У Win32 об'єкт синхронізації типу *ОС.К* реалізований через механізм

критичних секцій (critical section):

$K = ((\ К.захватити(\ T\ ))\ \longrightarrow\ K\ )$.

На жаль подія *К.захватити*(*T*) в Win32 реалізована за допомогою двох операцій `EnterCriticalSection()` і `LeaveCriticalSection()`. При такому підході функція `EnterCriticalSection()` дозволяє захопити критичну секцію і увійти до критичної ділянки, якщо вона вільна. Функція `LeaveCriticalSection()` інформує про вихід з критичної ділянки при його звільненні. Операція *T* в цьому випадку «ховається» між цими двома функціями:

```
EnterCriticalSection(K);
T; -- КД
LeaveCriticalSection(K);
```

Така схема вирішення задачі взаємного виключення менш надійна, оскільки відсутній зв'язок між конструкціями входу і виходу з критичної ділянки.

У мові Java *ОС.К* реалізований за допомогою синхронізованих блоків і подія *К.захватити*(*T*) реалізується таким чином.

```
object K;
 synchronized(K){
T;
 }
```

У мові C# об'єкт синхронізації типу *ОС.К* реалізований через механізм критичних секцій (critical section) і замків:

```
object K;
 lock(K){
 T;
}
```

Об'єкт *ОС.М.* Об'єкт синхронізації типу *ОС.М* реалізується за допомогою захищених модулів (`protected unit`), в яких використана концепція моніторів. Для реалізації в *ОС.М* дій *М.запис, М.обчислення, М.сигнал* використовуються захищені процедури, дії *М.читати* – захищені функції, дії *М.чекати* – захищені входи (рисунок 3.4).

Слід зазначити особливість використання захищених функцій, які призначені виключно для копіювання загальних змінних. Це дозволяє одночасне використання різними процесами захищених функцій, за рахунок чого скорочується час доступу до загальних змінних. Реалізація такої схеми доступу

можлива через динамічне створення копій загального ресурсу безпосередньо при копіюванні.

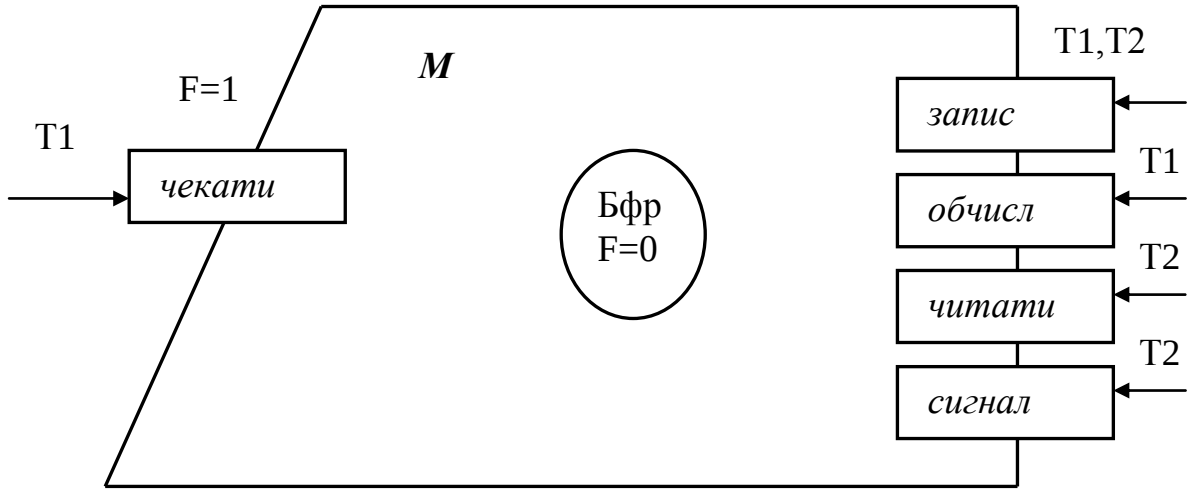


Рисунок 3.4 – Реалізація *ОК.М* за допомогою захищеного модуля

Нижче приведена реалізація об'єкту синхронізації типу *ОС.М* за допомогою захищеного модуля для вирішення задачі взаємного виключення і синхронізації, структура якого приведена на рисунку 3.4.

```
-- специфікація захищеного модуля
protected M is
procedure Запис(x: in integer);
procedure Обчислення(x,z: in integer; y: out integer);
procedure Сигнал;
function Читати return integer;
entry Чекати;
 private
Бфр: integer;
 F : integer:= 0;
end M;
 -- тіло захищеного модуля
protected M is
 procedure Запис(x: in integer)
 begin
 Бфр:= x;
 end;
 procedure Обчислення(x,z: in integer; y: out integer)is
 begin
 y:= x + y*Бфр;
 end;
 procedure Сигнал is
```

```
begin
F:= F+1;
end;
function Читати return integer is
begin
return Бфр;
end;
entry Чекати when F=1 is
begin
null;
end;
end M;
```

Вирішення задачі взаємного виключення забезпечується властивістю захищених операцій, характерних для моніторів, які гарантують виконання у будь-який момент часу тільки однієї моніторної процедури. Виняток становлять захищені функції, які в захищеному модулі можна виконувати одночасно.

У захищеному модулі *M* захищена функція забезпечує доступ по читанню до захищеного елементу `Бфр`, захищена процедура `Запис` дозволяють записувати нові значення для `Бфр`, а захищена процедура `Обчислення` – виконувати операції з `Бфр`.

Синхронізація реалізується в захищеному модулі *M* за допомогою захищеного входу `Чекати` і захищеної процедури `Сигнал`. Захищений вхід має спеціальну конструкцію – бар'єр, яка є логічним виразом, який перевіряється при кожному виклику захищеного входу. Якщо бар'єр приймає значення `true`, то вхід відкрити і виконується тіло захищеного модуля. Якщо бар'єр приймає значення `false`, то вхід відкрити і виконується тіло захищеного модуля.

Зміна значення бар'єру є сигналом про подію і виконується захищеною процедурою `Сигнал`, яка міняє значення прапора в бар'єрі.

Таким чином, захищений модуль дозволяє реалізувати об'єкт синхронізації типу *ОС.М,* причому оптимізувати при цьому доступ по читанню до загального ресурсу, розміщеного в *ОС.М* .

Реалізацію об'єкту синхронізації типу *ОС.М* можна також реалізувати засобами мови Java. Для цього будується клас, методи якого описуються за допомогою модифікатора `synchronized`. Операції *М.читати* і *М.чекати*

реалізуються за допомогою методів, в яких використовуються спеціальні методи `wait()` і `notify()`, що дозволяють блокувати і розблоковувати процеси (потоки) [12, 15].

3.3 Дослідження тупикових ситуації при використанні об'єктів комунікації і синхронізації

Розглянемо аналіз причин тупикових ситуації при використанні об'єктів комунікації і синхронізації в Аді.

Механізм рандеву. Механізм рандеву в мові Ада забезпечує взаємодію двох задач (див. рисунок 3.2). При цьому відповідно до специфікації входу передача даних може здійснюватися в будь-якому напрямі. Тому будемо розрізняти не приймаючу і передавальну задачі, а також задачі, що викликаються. На рисунку 3.2 такою, що викликається є задача *A*, а такою що викликає - задача *B,* в якій описаний вхід Data.

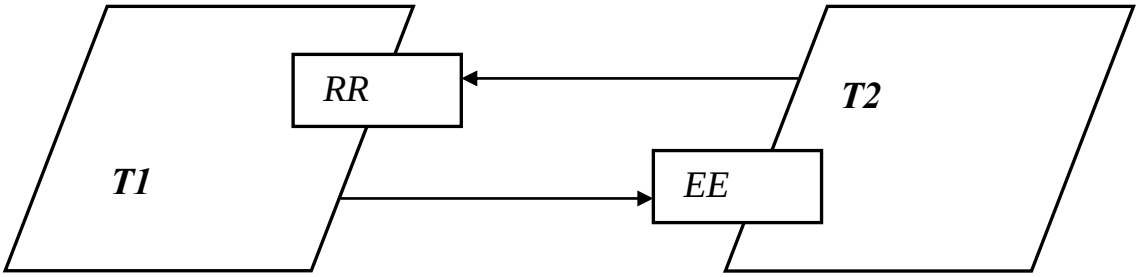
При взаємодії двох задач з використанням одного входу тупикова ситуація може виникнути у разі динамічного формування фактичних значень параметрів, коли, наприклад, розмір масиву, що передається не відповідає розміру, описаному в специфікації входу.

Причиною тупикової ситуації при взаємодії задач через рандеву може стати також наявність циклів на схемі взаємодії задач (рисунок 3.5) [13]. На рисунку 3.5 а) є прямий цикл, на рисунку 3.5 б) - непрямий цикл.

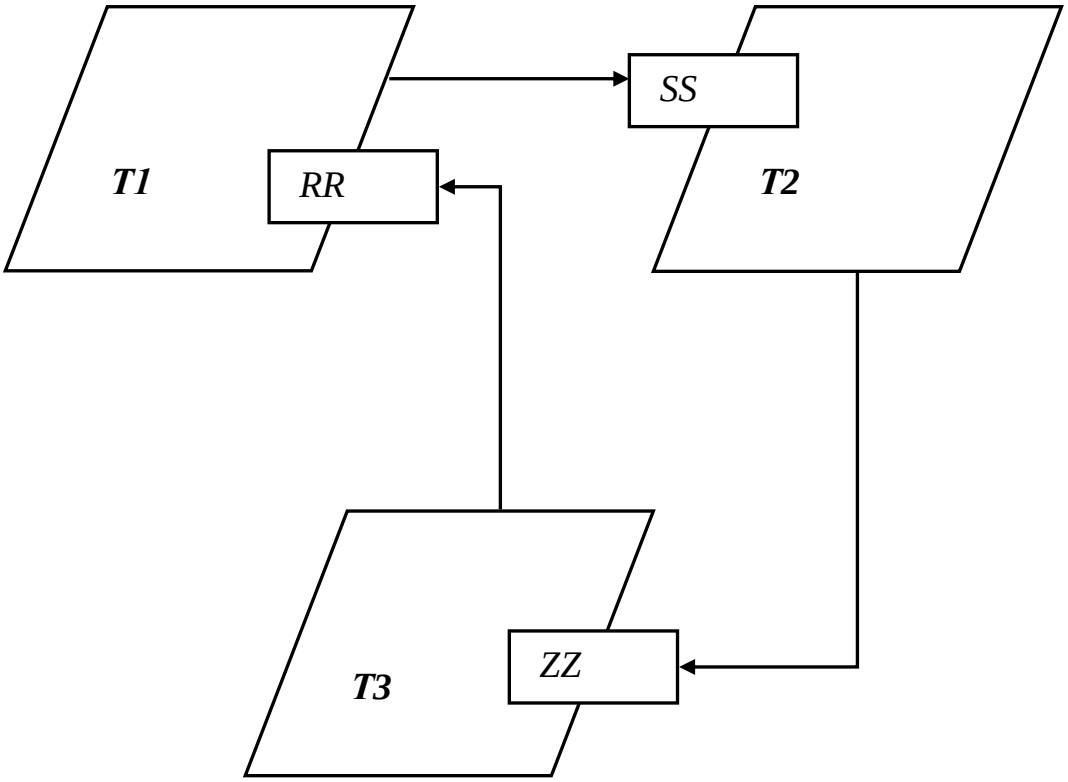
За наявності прямого циклу причина тупикової ситуації є порушення порядку виклику входів і порядку прийомів викликів входів в операторах `accept`. Нижче приведена програмна реалізація взаємодії задач *A* і *B* (рисунок 3.5 а)), що приводить до тупикової ситуації.

```
procedure PRG33 is
 task T1 is
 entry RR();
 end T1;
task T2 is
 entry EE();
 end T2;
```

```
 task body T1 is
 . . .
T2.EE(); -- виклик входу
accept RR(); -- прийом виклику входу
 . . .
end T1;
 task body T2 is
 . . .
T1.RR(); -- виклик входу
accept EE(); -- прийом виклику входу
 . . .
end T2;
 begin
 null;
 end PRG33;
```



а)



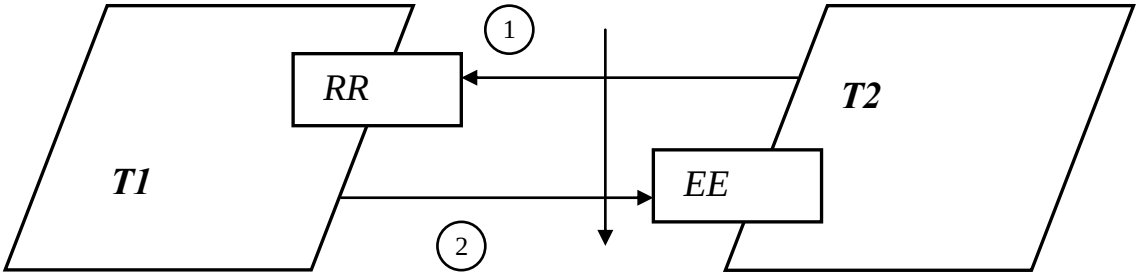
б)

Рисунок 3.5 – Тупикові ситуації в механізмі рандеву

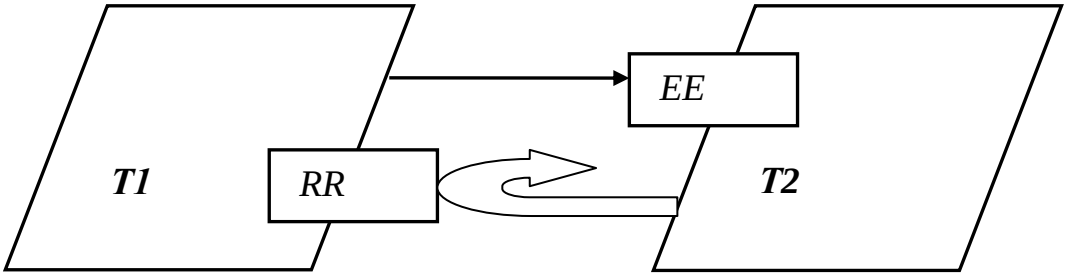
Задача *T1* блокується при виклику входу `T2.EE()`, а задача *T2* блокується при виклику входу `T1.RR()`. Це відбувається внаслідок того, що в задачах порушена послідовність викликів і прийомів викликів входів. Організація безтупикової взаємодії задач *T1* і *T2* можлива на основі:

- встановлення строго порядку викликів і прийомів викликів входів;
- використання умовних викликів входів;
- розміщенням входів в одній задачі;
- використанням буферизуючої задачі.

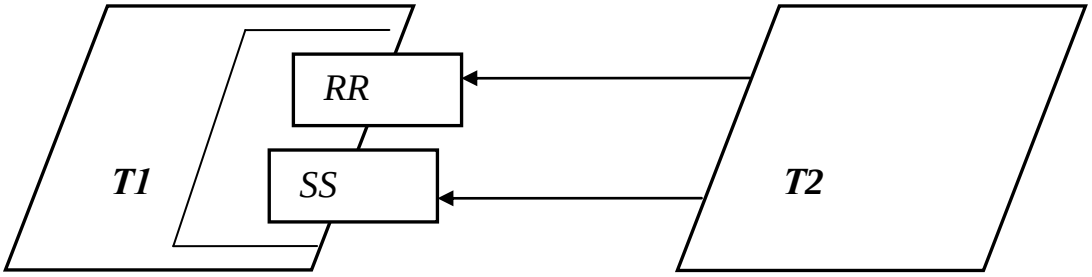
Схеми вказаних варіантів безтупикової взаємодії приведені на рисунках 3.6 і 3.7.



а)



б)



с)

Рисунок 3.6 – Безтупикові ситуації в механізмі рандеву

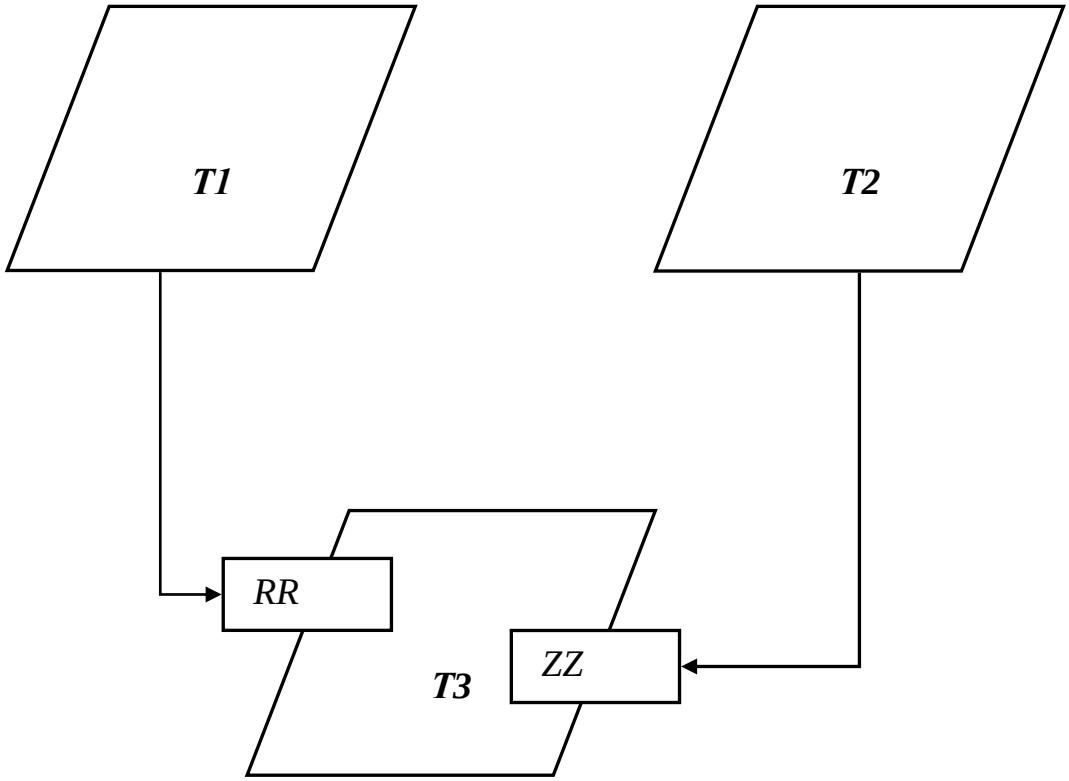


Рисунок 3.7 – Безтупикова ситуація в механізмі рандеву на основі буферизуючої задачі

Семафори. Семафори, реалізовані в мові Ада є об'єктами синхронізації типу *OC.S.* Причини виникнення безвиході для *OC.S* визначені в розділі 2.2. Проаналізуємо ці причини з врахуванням особливостей реалізації семафорів в Аді.

У Задачі взаємного виключення причинами безвиході є:

а) некоректна початкова установка семафора;

б) відсутність сигналу про вихід з критичної ділянки.

На рисунку 3.8 представлена схема вирішення задачі взаємного виключення за допомогою семафорів мови Ада.

```
------------------------------------------------------------
-- Ада.Семафори в задачі взаємного виключення --
------------------------------------------------------------
with Ada.Synchronous_Task_Control,Text_IO,Integer_Text_IO;
use Ada.Synchronous_Task_Control,Text_IO,Integer_Text_IO;
procedure PRG32 is

 Буфер: integer:= 10; -- загальний ресурс
  Sem_Ku: Suspension_Object; -- семафор
```

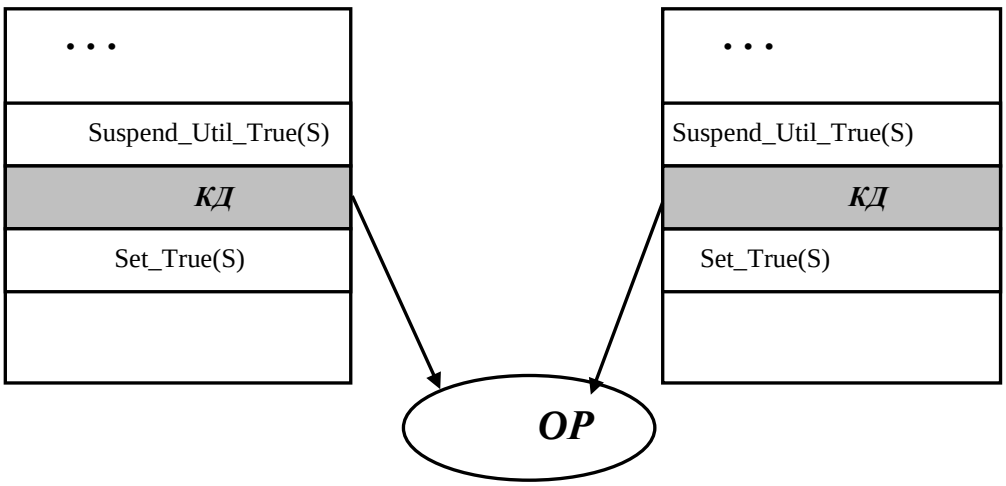


Рисунок 3.8 – Семафори в задачі взаємного виключення

```
 procedure Старт_задач is
task A;
task body A is
begin
 put_line("Process A started");
 -- Операція над загальним ресурсом
 Suspend_Until_True(Sem_Ku);
 Буфер := Буфер + 2; -- критична ділянка
 Set_True(Sem_Ku);
 put_line("Process A finished");
end A;

task B;
task body B is
begin
 put_line(" Process B started");
 -- Операція над загальним ресурсом
 Suspend_Until_True(Sem_Ku);
 Буфер:= Буфер - 25; -- критична ділянка
 Set_True(Sem_Ku);
 put_line(" Process B finished");
end B;
 begin
null;
 end Старт_задач;
-- основна процедура
begin
 put_line(" Main procedure started ");
```

```
Set_True(Sem_Ku); -- встановлення початкового семафора (true)
 Старт_задач; -- запуск задач A і B
end Lab37;
```

Захищені модулі. Захищені модулі, що реалізовують в мові Ада концепцію моніторів є об'єктами синхронізації типу *ОС.М.* Причини виникнення безвиході для *ОС.М* визначені в розділі 2.2. Проаналізуємо ці причини з врахуванням особливостей реалізації моніторів в Аді.

Безвихідь при використанні захищеного модуля для вирішення задачі взаємного виключення малоймовірна, оскільки доступ до загального ресурсу контролюється самим монітором і користувач не використовує явно дії, пов'язані із захопленням і звільненням загального ресурсу. Проблеми можуть виникати при специфічному використанні моніторів, наприклад, для вкладених моніторів, але таких ситуацій слід уникати. Захищені функції і захищені процедури забезпечують ефективне вирішення проблеми доступу до загального ресурсу. При використанні умовного доступу до загального ресурсу за допомогою захищених входів тупикова ситуація можлива у випадку, якщо інший процес не виконає зміну значення бар'єру у вході і заблокована задача залишиться в тупиковому стані.

Для задачі синхронізації, яка реалізується за допомогою виклику входу в задачі, що очікує і виклику процедури в сигналізуючій задачі, тупикова ситуація виникає при неможливості відкриття входу через відсутність зміни бар'єру або некоректності його зміни.

У захищеному модулі *М22* (рисунок 3.9) розглядається ситуація, коли процес *Т1* синхронізується з двома подіями, які матимуть місце в процесах *Т2* і *Т3*.

Вхід `Чекати` використовується процесом *Т1* для очікування двох сигналів, які поступлять від задач *Т2* і *Т2*. Реалізація сигналів від задач *Т2* і *Т3* здійснюється як виклик захищеної процедури `Сигнал`, яка міняє значення прапора. Помилкова реалізація процедури `Сигнал` не дозволить встановити прапор *F* в значення 2 і відкрити вхід `Чекати`. В цьому випадку задачу *Т1* буде заблоковано на вході `Чекати`.

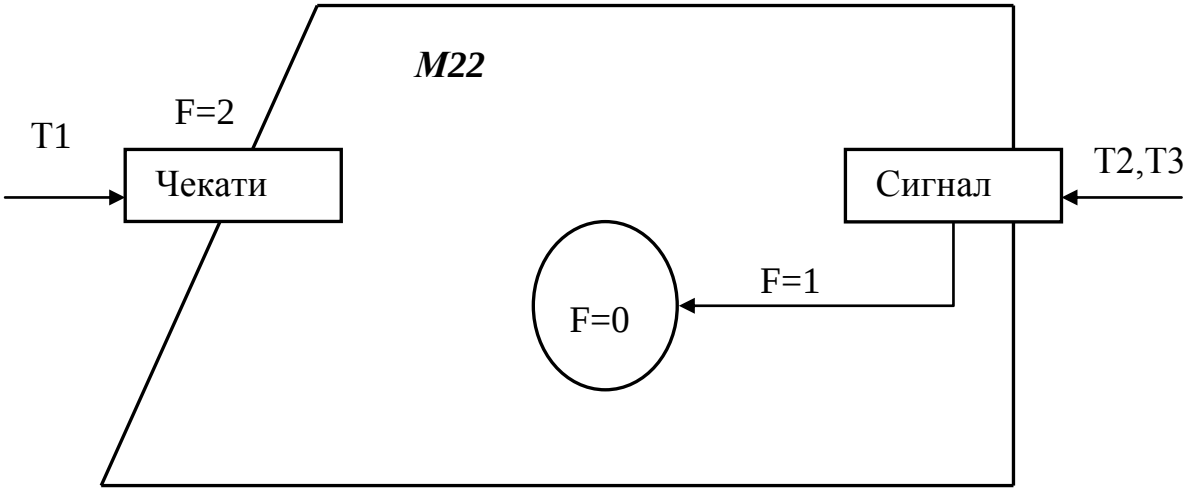


Рисунок 3.9 – Синхронізація за допомогою захищеного модуля

Уникнути тупикової ситуації можна при наступній реалізації процедури Сигнал:

```
procedure Сигнал is
begin
F:= F + 1;
end Сигнал;
```

В додатку А представлений лістинг програми для вирішення задачі множення матриці МА=МВ * МС в кластерній багатоядерній системі.

В додатку Б представлена довідка про використання результатів дипломної роботи.

У даному розділі розглянуті питання реалізації запропонованих моделей паралельних обчислень для КСБА. Реалізація виконується за допомогою спеціальних програмних засобів, наявних в сучасних мовах програмування і бібліотеках програмування. В якості базової мови розглядається мова програмування Ада.

ВИСНОВКИ

1. Розвиток сучасних кластерних систем пов’язаний зі збільшенням обчислювальної потужності вузлів системи, яке на сьогодні можна реалізувати шляхом використання у вузлах багатоядерних процесорів. Організація обчислень в КСБА пов’язана з вирішенням двох задач - організації ефективної взаємодії вузлів системи і обчислень в кожному вузлі, які у свою чергу вимагають ефективної реалізації обчислень в кожному ядрі і їх взаємодії. КСБА є складною системою, тому організація обчислювальних процесів і розробка програмного забезпечення для неї повинні базуватися на використанні математичних моделей паралельних обчислень, що забезпечують інтерфейс між архітектурою КСБА і моделлю програмування.

2. Виконаний аналіз особливостей організації обчислювальних процесів в КСБА. Показано, що система володіє двома рівнями паралельної обробки – між вузлами і на рівні кожного вузла і вимагає розробки моделей паралельних обчислень, що забезпечують опис поведінки процесів на обох рівнях.

3. Виконана розробка моделі обчислень КСБА1 для першого рівня. Розглянута можливість використання моделі КСБА1 для аналізу коректності взаємодії процесів на рівні вузлів. Показано, що модель дозволяє виявлення тупикових ситуацій. Сформульовано ряд правил, що забезпечують коректну взаємодію процесів при використанні об’єктів комунікації.

4. Запропонована математична модель другого рівня КСБА2, що описує поведінку процесів у вузлах КСБА. Модель враховує обидва способи організації взаємодії процесів, як основи загальних змінних, так і на основі передачі повідомлень. Виконаний аналіз можливості використання об’єктів комунікації для вирішення задач взаємного виключення і синхронізації процесів при використанні загальних змінних, який показав, що вони дозволяють ефективно вирішувати обидві задачі.

5. Розглянута можливість використання моделі КСБА2 для аналізу коректності взаємодії процесів у вузлах КСБА. Сформульовано ряд правил, що забезпечують коректну взаємодію процесів при використанні об’єктів

синхронізації.

6. Розглянута реалізація моделей обчислень КМСА1 і КСБА2. Показано, що реалізація моделей грунтується на використанні механізму процесів (потоків) і різних засобів взаємодії процесів, і може бути виконана засобами сучасних мов програмування і бібліотек. Запропонована реалізація об’єктів комунікації в моделі КСБА1 за допомогою виклику віддалених процедур (RPC механізму), які дозволяють організувати передачу повідомлень між вузлами системи з врахуванням особливостей конкретної взаємодії.

7. Запропонована реалізація об’єктів комунікації в моделі КСБА2, заснована на механізмі рандеву. Показано, що механізм рандеву дозволяє реалізувати всі складові об’єкту комунікації, окрім буферизації, яку можна додатково реалізувати через буферизуючий процес.

8. Виконаний аналіз реалізації об’єктів синхронізації, який показав, що використовувані в сучасних мовах і бібліотеках засоби організації взаємодії процесів на основі моделі загальних змінних дозволяють використовувати їх для реалізації об’єктів комунікації всіх трьох видів і з їх допомогою вирішувати задачі взаємного виключення і синхронізації.

9. Розглянуті причини виникнення тупикових ситуацій при реалізації об’єктів синхронізації і комунікації в моделях КСБА1 і КСБА2. Запропоновані варіанти використання об’єктів комунікації і синхронізації, що дозволяють у ряді випадків уникнути тупикових ситуацій при взаємодії процесів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тютюнник М.І. Паралельні алгоритми та засоби для розв’язання деяких задач масових обчислень [Електронний ресурс]. – Режим доступу: http://www.iapmm.lviv.ua/chyt2010/materials/pc2010-02-T-32.pdf.
2. Воеводин Вл.В., Жуматий С.А. Вычислительное дело и кластерные системы. – М.: Изд-во МГУ, 2007. – 150 с.
3. Гофф Макс К. Сетевые распределенные вычисления. Достижения и проблемы. – М.: Кудиц-Образ, 2005. – 320 с.
4. Эндрюс Г. Основы многопоточного, параллельного и распределенного программирования.: Пер. с англ. – М.: Изд. дом Вильямс. – 2003. – 512 с.
5. Кулаков Ю.О., Луцький Г.М. Комп’ютерні мережі. – К.: Юніор, 2003. – 400 с.
6. Список 500 самых мощных компьютеров мира. 31-я редакция [Електронний ресурс]. – Режим доступу: http:// www.parallel.ru.
7. Каляев И.А., Левин И.И., Семерников Е.А., Шмойлов В.И. Реконфигурируемые мультиконвейерные вычислительные структуры. – М.: ид-во ЮНЦ РАН, 2008. – 320 с.
8. Андрианов С.Н., Дегтярев А.Б. Параллельные и распределенные вычисления. Часть1. – СПб. : Изд–во С– Петерб. унив.- та, 2007. – 61 с.
9. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие. – М.: Изд-во МГУ, 2009. – 77 с.
10. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ. – Петербург, 2002. – 608 с.
11. Гергель В.П. Теория и практика параллельных вычислений. – М.: Бином. Лаборатория знаний, 2007. – 424 с.
12. Жуков І.А., Корочків О.В. Паралельні та розподілені обчислення: Навч. посібник. – К.: Корнейчук, 2007. – 246 с.
13. Бар Р. Язык Ада в проектировании систем. – М.: Мир, 1988. – 320 с.
14. Василеску Ю. Прикладное программирование на языке Ада 95. – М.: Мир, 1990. – 332 с.
15. Ноутон П., Шилдт Г. Java2: Пер. с англ. – СПб.: БХВ – Петербург, 2000. –

1072 с.

16. Симкин С., Барлетт Н., Лесли А. Программирование на Java. Путеводитель. – К.: НИПФ ДиаСофт Лтд., 1996. – 736 с.
17. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Изд-во МГУ, 2004. – 71 с.
18. Корочкин А., Тулинов А. Экспериментальный анализ эффективности PVM и MPI в распределенных вычислительных системах // Вісн. НТУУ “КПІ”, Інформатика, управління та обчислювальна техніка. – 2002. – № 38. – С. 154 – 158.
19. Kyung-Ah Chang, Byung-Rae Lee, Tai-Yun Kim. Authentication Service Model Supporting Multiple Domains in Distributed Computing // Proceedings of the International Conference on Computational Sciences-Part I. - Vol. 2073. – 2001. - P. 413 – 422.
20. Водяхо А.И., Горнец Н.Н., Пузанков Д.В. Высокопроизводительные системы обработки данных. – М.: Мир, 1997. – 232 с.
21. Hoare C.A.R. Communicating Sequential Processes // Communications of ACM. - Vol. 21. – № 8. – 1978. – P. 666 – 667.
22. Джоунз Г. Программирование на языке ОККАМ. – М.: Мир, 1989. – 246 с.
23. El – Rewini H, Lewis T. Distributed and Parallel Computing. – Manning Pub. Co., 1998. – 430 p.
24. Burns A., Wellings A. Real-Time Systems and Programming Languages. – Addison – Wesley, 2001. – 386 p.
25. Воеводин В.В. Математические модели и методы в параллельных процессах. – М.: Наука, 1984. – 296 с.
26. Дейтел Д. Введение в операционные системы. – М.: Мир, – 1989. – 360 с.
27. Burns A., Wellings A. Concurrency in Ada. – Cambridge: Cambridge University Press, 1995. – 420 p.
28. Любченко В.С. К проблеме создания модели параллельных вычислений // Материалы 3-го межд. Семинара «Параллельные вычисления и задачи управления». – Самара. - 2006. – С. 181 – 186.
29. Воеводин В.В. Математические модели и методы в параллельных процесах. –

М.: Наука, 1984. – 296 с.

30. Топорков В.В. Модели распределенных вычислений. – М.: ФИЗМАТЛИТ, 2004. – 320 с.
31. Любченко В.С. Автоматная модель параллельных вычислений // Труды 3-й межд. конф. «Высокопроизводительные параллельные вычисления на кластерных системах». – М. - 2006. – С. 1359 – 1374.
32. Хоар Ч. Взаимодействующие последовательные процессы. – М.: Мир, 1989. – 180 с.
33. Юткин А. Объектные технологии в распределенных системах // Открытые системы. – № 3(11). – 1995. – С. 23–28.
34. Hoare C.A.R. Monitors: An Operating System Structuring Concept // Communications of ACM. – Vol.17. - № 10. - 1974. – P. 549 -557.
35. Корочкин А., Жужель М., Авдеев А., Корочкин Д. Защищенный модуль как универсальное средство синхронизации процес сов // Вісн. НТУУ “КПІ”, Інформатика, управління та обчислювальна техніка. – 2001. – № 34. – С. 137 – 145.
36. Reference Manuel for the Ada Programming Language. – United States Department of Defense. – 2005. – 466 p.
37. Соловьев Г.Н., Никитин В.Д. Операционные системы ЭВМ. – М.: Высш. школа, 1989. – 255 с.
38. Троелсон Е. C# и платформа.NET. Библиотека программиста. – СПб.: Питер, 2004. – 796 с.
39. Методичні рекомендації до виконання дипломної роботи з освітньо-кваліфікаційного рівня “Магістр”. Спеціальність „Комп’ютерні системи та мережі” / О.М. Березький, Р.Б. Трембач, Г.М. Мельник / Під ред. О.М. Березького – Тернопіль: ТНЕУ, 2012.– 42 с.

Додаток А

Лістинг програми для вирішення задачі множення матриці в кластерній багатоядерній системі

```
With Ada.Text_IO, Ada.Integer_Text_IO;
Use  Ada.Text_IO, Ada.Integer_Text_IO;
package DDD is
   N: integer := 1200;
   P: integer :=  6;
   H: integer := N/ (P * 4);
   type Vector is array (1 .. N) of integer;
   type Matrix is array (integer range <>) of Vector;
   subtype MatrixN is Matrix (1 .. N);
   subtype Matrix4H is Matrix (1 .. 4*H);
   subtype MatrixH  is Matrix (1 .. H);
end DDD;

-- захищений модуль для клієнта
protected ZZZ is
   procedure WriteMB(MM: in MatrixN);
   function ReadMC return MatrixN;
   entry Wait1;
   entry Wait2;
   procedure Signal1;
   procedure Signal2;
private
   MC: MatrixN;
   F1: integer:=0;
   F2: integer:=0;
end ZZZ;

protected  body ZZZ is

   procedure WriteMB(MM: in MatrixN) is
   begin
      NC:= MM;
   end WriteMB;
   function ReadMC return MatrixN is
   begin
      return MC;
   end ReadMC;
   entry Wait1 when F1= 1 is
   begin
      null;
   end Wait1;
   entry Wait2 is
   begin
      null;
   end Wait2;
   procedure Signal1  is
   begin
      F1: = 1;
   end Signal1;
   procedure Signal2 is
   begin
      F2:= F + 1;
   end Signal1;
end ZZZ;
-- Клієнт
package Data is
   pragma Remote_Procedure_Call;
   procedure Culc(MB: in Matrix4H; MC: in MatrixN;
```

```
                                          MA: out Matrix4H);
end Data;

package  body Data is
   procedure Culc(MB: in Matrix4H; MC: in MatrixN;
                                           MA: out Matrix4H) is
      task P1;
      task P2;
      task P3;
      task P4;
      task body P1 is
         MC1: MatrixN;
      begin
         -- чекати отримання даних від сервера
         ZZZ.Wait1;
         -- копия MC
         MC1:= ZZZ.ReadMC;
         -- счет
         for i in 1 .. H loop
            for j in  1 .. N loop
               MA(i)(j):= 0;
               for k in 1.. N loop
                  MA(i)(j):= MA(i)(j)+MB(i)(k)*MC1(k)(j);
               end loop;
            end loop;
         end loop;
         --  сигнал про завершення обчислення
         ZZZ.Signal2;
      end P1;
      task P2  is
         MC2: MatrixN;
      begin
         -- чекати отримання даних від сервера
         ZZZ.Wait1;
         -- копия MC
         MC2:= ZZZ.ReadMC;
         -- обчислення
         for i in H+1 .. 2*H loop
            for j in  1 .. N loop
               MA(i)(j):= 0;
               for k in 1.. N loop
                  MA(i)(j):= MA(i)(j)+MB(i)(k)*MC2(k)(j);
               end loop;
            end loop;
         end loop;
         --  сигнал про завершення обчислення
         ZZZ.Signal2;

      end P2;
      task P3  is
         MC3: MatrixN;
      begin
         -- чекати отримання даних від сервера
         ZZZ.Wait1;
         -- копия MC
         MC3:= ZZZ.ReadMC;
         -- счет
         for i in 2*H+1 .. 3*H loop
            for j in  1 .. N loop
               MA(i)(j):= 0;
               for k in 1.. N loop
                  MA(i)(j):= MA(i)(j)+MB(i)(k)*MC3(k)(j);
               end loop;
            end loop;
         end loop;
         --  сигнал про завершення обчислення
```

```
         ZZZ.Signal2;
      end P3;
      task P4 is
         MC4: MatrixN;
      begin
         -- чекати отримання даних від сервера
         ZZZ.Wait1;
         -- копія MC
         MC4:= ZZZ.ReadMC;
         -- обчислення
         for i in 3*H+1 .. N loop
            for j in  1 .. N loop
               MA(i)(j):= 0;
               for k in 1.. N loop
                  MA(i)(j):= MA(i)(j)+MB(i)(k)*MC1(k)(j);
               end loop;
            end loop;
         end loop;
         --  сигнал про завершення обчислення
         ZZZ.Signal2;
      end P4;
   begin
      ZZZ.Signal1;
   end Culc;
begin
   null;
end Data;
-- Сервер. Захищений модуль --------
protected SSS is
   procedure WriteMB(MM: in MatrixN);
   function ReadMC return MatrixN;
   entry Wait1;
   entry Wait2;
   procedure Signal1;
   procedure Signal2;
private
   MC: MatrixN;
   F1: integer:=0;
   F2: integer:=0;
end SSS;
protected  body SSS is
   procedure WriteMB(MM: in MatrixN) is
   begin
      NC:= MM;
   end WriteMB;
   function ReadMC return MatrixN is
   begin
      return MC;
   end ReadMC;
   entry Wait1 when F1= 1 is
   begin
      null;
   end Wait1;
   entry Wait2 is
   begin
      null;
   end Wait2;
   procedure Signal1  is
   begin
      F1: = 1;
   end Signal1;
   procedure Signal2 is
   begin
      F2:= F + 1;
   end Signal1;
end SSS;
```

```
-- Сервер---------------------------------
procedure Work is
   MA, MB, MC: MatrixN;
   task P1;
   task body P1 is
      MC1: MatrixN;
   begin
      -- чекати вводу даних
      SSS.Wait1;
      -- копія MC
      MC1:= SSS.ReadMC;
      -- виклик віддаленої процедури
      Data.Culc(MB(1..4*H), MC1, MA(1..4*H));
      --  сигнал про завершення обчислення
      SSS.Signal2;
   end P1;

   task P2;
   task body P2 is
      MC1: MatrixN;
   begin
      -- чекати вводу даних
      SSS.Wait1;
      -- копія MC
      MC1:= SSS.ReadMC;
      -- виклик віддаленої процедури
      Data.Culc(MB(4*H+1..8*H), MC1, MA(4*H+1..8*H));
      --  сигнал про завершення обчислення
      SSS.Signal2;
   end P2;

   task P3;
   task body P3 is
      MC1: MatrixN;
   begin
      -- чекати вводу даних
      SSS.Wait1;
      -- копія MC
      MC1:= SSS.ReadMC;
      -- виклик віддаленої процедури
      Data.Culc(MB(8*H+1..12*H), MC1, MA(8*H+1..12*H));
      --  сигнал про завершення обчислення
      SSS.Signal2;
   end P3;

   task P4;
   task body P4 is
      MC1: MatrixN;
   begin
      -- чекати вводу даних
      SSS.Wait1;
      -- копія MC
      MC1:= SSS.ReadMC;
      -- виклик віддаленої процедури
      Data.Culc(MB(12*H+1..16*H), MC1, MA(12*H+1..16*H));
      --  сигнал про завершення обчислення
      SSS.Signal2;
   end P4;

   task P5;
   task body P5 is
      MC1: MatrixN;
   begin
      -- чекати вводу даних
      SSS.Wait1;
```

```
      -- копія MC
      MC1:= SSS.ReadMC;
      -- виклик віддаленої процедури
      Data.Culc(MB(16*H+1..20*H), MC1, MA(16*H+1..20*H));
      --  сигнал про завершення обчислення
      SSS.Signal2;
   end P5;

   task P6;
   task body P6 is
      MC1: MatrixN;
   begin
      -- чекати вводу даних
      SSS.Wait1;
      -- копія MC
      MC1:= SSS.ReadMC;
      -- виклик віддаленої процедури
      Data.Culc(MB(20*H+1..N), MC1, MA(20*H+1..N));
      --  сигнал про завершення обчислення
      SSS.Signal2;
   end P6;

-- основна процедура
begin
   --  ввід даних
   for i in 3*H+1 .. N loop
      for j in  1 .. N loop
         get(MB(i)(j));
         get(MC(i)(j));
      end loop;
    end loop;

   -- сигнал про ввід
   ZZZ.Sinal1;

   -- чекати завершення обчислення
   ZZZ.Wait2;

   -- вивід результату
   for i in 3*H+1 .. N loop
      for j in  1 .. N loop
         put(MA(i)(j));
      end loop;
    end loop;

 end Work;
```

Додаток Б

Довідка про використання