5. Автоматические подводные аппараты / М.Д. Агеев, Б.А. Касаткин, Л.В. Киселев и др. – Л.: Судостроение, 1981 (Техника освоения океана). – 224 с.
6. Принципы построения технических средств исследования океана / Отв. ред. В.С.Ястребов. – М.: Наука, 1992. – 325 с.
7. Архангельский В.И., Богаенко И.Н., Грабовский Г.Г., Рюмшин Н.А. Системы фуцци-управления. К.: Техника, 1997. – 208 с.
8. Кондратенко Ю., Аль Зубі І., Гаращенко О. Аналіз процесів проектування та шляхів підвищення ефективності цифрових регуляторів нечітких систем управління // Вісник Національного університету "Львівська політехніка" "Комп'ютерна інженерія та інформаційні технології", 2002, – №468. – С.83-90.
9. Блінцов С.В. Синтез нечіткого регулятора для керування вертикальною складовою руху підводного апарата // Автоматика-2003: Матеріали 10-ї міжнародної конференції по автоматичному управлінню, м. Севастополь, 15-19 веревня 2003 р.: в 3-х т. – Севастополь: Вид-во СевНТУ, 2003. – Т. 2. – С. 119-120.
10. Справочник по теории корабля / Под ред. Я.И.Войткунского. Том I. – Л.: Судостроение, 1985. – 768 с.
11. Гостев В.И. Синтез нечетких регуляторов систем автоматического управления. – К.: Радіоаматор, 2003. – 512 с.
12. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика. – М.: Горячая линия - Телеком, 2002. – 382 с.

Б. Демида, Д. Пелешко, М. Пасєка, Б. Садов'як, О. Маханов, Д. Вачаєв
Національний університет "Львівська політехніка"

УДК 681.142.2

ПРОЕКТУВАННЯ БАГАТОРІВНЕВОЇ ІНФОРМАЦІЙНОЇ ПЛАТФОРМИ ДЛЯ РОЗПОДІЛЕНИХ БІЗНЕС-СИСТЕМ

Пропонується проект багаторівневої платформи для інформаційно-аналітичної розподіленої комп'ютерної системи управління та ведення бізнес-діяльності. Проведено декомпозицію системи. Детально розроблено базовий сервер системи.

There is proposed project of five level platform for information, analytical distributed, computer system for control and management of big business. The decomposition of proposed system is realized. The base system server is designed in detail.

Вступ

Сучасні інформаційні технології стають все більше орієнтованими на забезпечення ефективного середовища для виконання бізнес-процесів корпорацій. Інформаційна система поступово перетворюється зі зручного інструмента в невід'ємне сере-

довище для виконання робіт. Працівники в такому середовищі концентруються не на інструментах досягнення мети, а на самій меті. Вони стають ближчими до прикладних задач і значною мірою абстрагуються від інформаційних технологій.

У роботі розглянуто аспекти змін в організації сучасного бізнесу, в світлі яких формулюються нові вимоги до ефективного функціонування загального комунікаційно-обчислювального середовища.

В основній частині роботи сформульовано ряд принципів побудови розподіленої інформаційної системи і наведено приклад застосування сучасних підходів для створення таких систем.

Аналіз існуючих обчислювальних систем

Сьогодні практично усі великі інформаційні системи є розподіленими. Розподіленою називається така система, в якій обробка інформації зосереджена не на одній обчислювальній машині, а розподілена між декількома комп'ютерами.

Оскільки в наші дні такі системи все ширше застосовуються, розробники ПЗ повинні бути знайомі з особливостями їхнього проектування. Донедавна усі великі системи в основному були централізованими і запускалися на одній головній обчислювальній машині (мейнфреймі) з під'єднаними до неї терміналами. Термінали практично не займалися обробкою інформації – обробка усієї логіки здійснювалась на головній машині. Розробникам таких систем не доводилося займатися проблемами розподілених обчислень.

Сучасні системи – це рознесення логіки (децентралізація) на різні машини. У результаті зростає швидкість обробки даних, проте зростає і складність таких систем. З'явились завдання, яких просто не існувало у першій моделі обчислень. Отже, зросли вимоги до фахового проектування систем.

Пропонована стаття є прикладом побудови багаторівневих розподілених обчислювальних систем, зокрема розглянуто п'ятирівневу систему.

Усі сучасні програмні системи можна розділити на три великі класи:

1. Прикладні програмні системи, призначені для роботи лише на одному персональному комп'ютері або робочій станції. До них належать текстові процесори, електронні таблиці, графічні системи тощо.
2. Вбудовані системи, призначені для роботи на одному процесорі або на інтегрованій групі процесорів. До них належать системи керування побутовими пристроями, різними приладами тощо.
3. Розподілені системи, в яких програмне забезпечення виконується на слабо інтегрованій групі паралельно працюючих процесорів, зв'язаних між собою мережею. До них належать системи банкоматів будь-якого банку, видавничі системи, системи ПЗ колективного користування та ін.

Сьогодні між перерахованими класами програмних систем існують чіткі межі, які надалі будуть все більше нівелюватись. У майбутньому, коли високошвидкісні безпровідні мережі стануть широкодоступними, з'явиться можливість динамічно інтегрувати пристрої з вбудованими програмними системами, наприклад, електронні органайзери з більш загальними системами.

Характеристика об'єкта дослідження

У [1] виділено шість основних характеристик розподілених систем.

1. *Спільне використання ресурсів.* Розподілені системи допускають спільне використання апаратних і програмних ресурсів, наприклад, жорстких дисків, принтерів, файлів, компіляторів тощо, зв'язаних за допомогою мережі. Очевидно, що розподіл ресурсів можливий також у багатокористувацьких системах, але в цьому випадку за надання ресурсів та керування ними повинен відповідати центральний комп'ютер.
2. *Відкритість.* Це можливість розширювати систему шляхом додавання нових ресурсів. Розподілені системи – це відкриті системи, до яких під'єднують апаратне і програмне забезпечення різних виробників, котрі відповідають стандартам *OSI*.
3. *Паралельність.* У розподілених системах кілька процесів можуть виконуватися одночасно на різних комп'ютерах в мережі. Ці процеси можуть (але не обов'язково) взаємодіяти один з одним під час їхнього виконання.
4. *Масштабованість.* Усі розподілені системи масштабуються: щоб система відповідала новим вимогам, її можна нарощувати за допомогою додавання нових обчислювальних ресурсів. Проте загалом нарощування обмежується лише збільшенням пропускної здатності мережі, яка об'єднує окремі комп'ютери системи.
5. *Стійкість до збоїв у системі.* Наявність декількох комп'ютерів і можливість дублювання інформації означає, що розподілені системи є стійкі до визначених апаратних і програмних помилок. Більшість розподілених систем у випадку помилки, як правило, можуть зберігати хоча б часткову функціональність (дієздатність).
6. *Прозорість.* Ця властивість означає, що користувачам наданий повністю прозорий доступ до ресурсів і водночас від них прихована інформація про розподіл ресурсів у системі. Однак у багатьох випадках конкретні знання про організацію системи допомагають користувачу краще використовувати ресурси.

Очевидно, що розподіленим системам властиві деякі недоліки:

- *Складність.* Розподілені системи складніші ніж централізовані. Набагато важче зрозуміти й оцінити властивості розподілених систем загалом, а також тестувати ці системи.
- *Безпека.* Зазвичай доступ до системи можна одержати з декількох різних машин, повідомлення в мережі можуть переглядатися або перехоплюватися. Тому у розподіленій системі складніше реалізувати систему безпеки.
- *Управління.* Система може складатися з різнотипних комп'ютерів, на яких можуть бути встановлені різні версії операційних систем. Помилки на одній машині можуть поширюватися на інші машини з непередбачуваними наслідками. Тому потрібно значно більше зусиль, щоб керувати і підтримувати систему в робочому стані.

При розгляді переваг і недоліків розподілених систем у [2] визначається ряд критичних проблем проектування таких систем (таблиця).

Проблеми проектування розподілених систем

Проблема проектування	Опис
Ідентифікація ресурсів	Ресурси в розподіленій системі розміщуються на різних комп'ютерах, тому систему імен ресурсів треба продумати так, щоб користувачі могли без надмірних зусиль відкривати необхідні ресурси і посилатися на них. Прикладом може бути система уніфікованого покажчика ресурсів *URL*, яка визначає адреси *Web*-сторінок.
Комунікації	Універсальна дієвість *Internet* і ефективна реалізація протоколів *TCP/IP* в *Internet* для більшості розподілених систем є прикладом найбільш ефективного способу організації взаємодії між комп'ютерами. Однак там, де на продуктивність, надійність накладаються спеціальні вимоги, можна скористатися альтернативними способами системних комунікацій.
Якість системного сервісу	Якість запропонованого системою сервісу відображає її продуктивність, працездатність і надійність. На якість сервісу впливає цілий ряд факторів: розподіл системних процесів, розподіл ресурсів, системні та мережні апаратні засоби і можливості адаптації системи.
Архітектура програмного забезпечення	Архітектура програмного забезпечення описує розподіл системних функцій по компонентах системи, а також розподіл цих компонентів по процесорах. Якщо треба підтримувати високу якість системного сервісу, вибір правильної архітектури є вирішальним фактором.

Огляд існуючих архітектур

Завдання розробників розподілених систем – запроектувати програмне або апаратне забезпечення так, щоб забезпечити всі описані характеристики розподіленої системи. А для цього треба знати переваги і недоліки різних архітектур розподілених систем. Розглянемо три типи архітектур розподілених систем.

1. *Багатопроцесорна архітектура*. Найпростіша розподілена система. Характеризується розподілом виконання скінченного набору процесів між скінченною кількістю процесорів.
2. *Мікроядерна розподілена архітектура з можливістю адаптації*. У такій архітектурі прикладні програми використовують локальні ресурси як кеш для глобальних мережевих ресурсів [6].
3. *Архітектура клієнт/сервер*. У цій моделі систему можна представити як набір сервісів, що надаються клієнтам серверами. У таких системах сервери і клієнти суттєво відрізняються один від одного.
4. *Архітектура розподілених об'єктів*. У цьому випадку між серверами і клієнтами немає значних відмінностей і систему можна подати як набір взаємодіючих об'єктів, розташування яких не має принципового значення. Між постачальником сервісів і їх користувачами не існує відмінностей.

У розподіленій системі різні системні компоненти можуть бути реалізовані на різних мовах програмування і виконуватися на різних типах процесорів. Моделі даних, подання інформації і протоколи взаємодії – все це не обов'язково має бути однотипним у розподіленій системі. Отже, для розподілених систем потрібне таке програмне забезпечення, яке могло б керувати цими різнотипними частинами і гарантувати взаємодію й обмін даними між ними. Проміжне програмне забезпечення належить саме до такого класу ПЗ. Воно знаходиться посередині між різними частинами розподілених компонентів системи.

Існують різні типи проміжного ПЗ, яке може підтримувати розподілені обчислення. Як правило, таке ПЗ складається з готових компонентів і не вимагає від розробників спеціальних доробок. Прикладами проміжного ПЗ можуть бути програми керування взаємодією з базами даних, менеджери транзакцій, перетворювачі даних, комунікаційні інспектори та ін.

Розподілені системи зазвичай розробляються на основі об'єктно-орієнтованого підходу. Ці системи створюються з слабо інтегрованих частин, кожна з яких може безпосередньо взаємодіяти як з користувачем, так і з іншими частинами системи. Ці частини, якщо це можливо, повинні реагувати на незалежні події. Програмні об'єкти, побудовані на основі таких принципів, є природними компонентами розподілених систем.

Розглянемо кожну архітектуру більш детально.

Багатопроцесорна архітектура

Найпростішою розподіленою системою є багатопроцесорна система. Вона складається з набору різних процесів, які можуть (але не обов'язково) виконуватися на різних процесорах. Ця модель часто використовується у великих системах реального часу. Ці системи збирають інформацію, приймають на її основі рішення і відправляють сигнали (повідомлення) виконавчому механізму, який змінює системне оточення. В принципі всі процеси, які пов'язані із збиранням інформації, прийняттям рішень і керуванням виконавчим механізмом, можуть виконуватися на одному процесорі під керуванням основного планувальника завдань. Використання декількох процесорів підвищує продуктивність системи та її здатність до відновлення. Процеси можуть перерозподілятися між процесорами і керуватися диспетчером процесів.

Системи ПЗ, які одночасно виконують певну скінченну кількість процесів, необов'язково є розподіленими. Якщо в системі є більше ніж один процесор, реалізувати розподіл процесів не важко. Однак при створенні багатопроцесорних програмних систем необов'язково відштовхуватися тільки від розподілених систем. При проектуванні систем такого типу, власне кажучи, використовується аналогічний підхід, як і при проектуванні систем реального часу.

Мікроядерна розподілена архітектура (МЯРА)

Відповідно до філософії МЯРА можна створити середовище, в якому робота виконується або мінімальною реалізацією операційної системи, або гнучким програмним забезпеченням проміжного рівня. Обидва рівні кооперуються для підтримки розподіленої об'єктної системи, яка добре конфігурується. Так можуть бути ефективно вирішені потреби визначених застосувань і декілька моделей можуть співіснувати.

Прикладні системи, які можуть виконуватися за допомогою такої системи, не тільки отримують очевидні переваги за рахунок високоефективної мікроядерної архітектури, але й можуть виконуватися і взаємодіяти у межах традиційних середовищ, таких як *Windows* і ін. Сервісні можливості, які надає мінімальне мікроядро, можуть бути легко перенесені на комерційні операційні системи. Це може бути досягнено або за допомогою модифікації комерційних операційних систем і відповідного експортування підмножини фізичних ресурсів або за допомогою реалізації відповідного адаптаційного рівня.

Архітектура клієнт/сервер

В архітектурі клієнт/сервер програмний додаток подається як набір сервісів, які надаються серверами, і множиною клієнтів, які використовують ці сервіси [1]. Клієнти повинні знати про доступні (наявні) сервери і можуть не знати про існування інших

клієнтів. У загальному випадку, розрізняючи клієнтів і сервери, треба мати на увазі скоріше логічні процеси, ніж фізичні машини, на яких виконуються ці процеси.

Архітектура системи клієнт/сервер повинна відтворювати логічну структуру розроблюваного програмного продукту. Характерним для неї є структурування програмного забезпечення на три рівні:

- Рівень представлення забезпечує надання інформації для користувачів і взаємодію з ними.
- Рівень виконання реалізує логіку роботи додатка.
- Рівень управління даними, на якому виконуються всі операції з базами даних.

У централізованих системах між цими рівнями не існує чіткого поділу. Однак при проектуванні розподілених систем необхідно розділяти ці рівні, з метою потреби чіткого розмежування кожного рівня на різних комп'ютерах.

Найпростішою архітектурою клієнт/сервер є дворівнева, в якій програма складається із сервера (або множини ідентичних серверів) і групи клієнтів. Існує два види такої архітектури (рис.1).

1. *Модель тонкого клієнта*. У цій моделі вся робота програми і керування даними виконується на сервері. На клієнтській машині запускається тільки ПЗ рівня представлення.
2. *Модель товстого клієнта*. В цій моделі сервер тільки керує даними. На клієнтській машині реалізована робота програми і взаємодія з користувачем системи.

Тонкий клієнт дворівневої архітектури - найпростіший спосіб переведення існуючих централізованих систем в архітектуру клієнт/сервер. Користувацький інтерфейс у цих системах переміщується на персональний комп'ютер, а сама програма виконує функції сервера, тобто виконує всі процеси додатка і керує даними. Модель тонкого клієнта можна також реалізувати там, де клієнти є звичайними мережними пристроями, а не персональними комп'ютерами або робочими станціями. Мережні пристрої запускають *Internet*-броузер і користувацький інтерфейс, який реалізований всередині системи. Головний недолік моделі тонкого клієнта – велика завантаженість сервера. Всі обчислення виконуються на сервері, а це може призвести до значного мережного трафіка між клієнтом і сервером та завантаження сервера. До того ж сучасні комп'ютери мають достатню обчислювальну потужність, але вона практично не використовується в моделі тонкого клієнта.

Рис.1. Моделі тонкого і товстого клієнтів

На противагу тонкому модель товстого клієнта використовує обчислювальну потужність локальних машин: і рівень виконання додатка, і рівень представлення містяться на комп'ютері клієнта. Сервер тут є тільки сервером транзакцій, який керує всіма транзакціями до баз даних (БД).

Оскільки в моделі товстого клієнта виконання програмного додатка організовано більш ефективно, ніж у моделі тонкого клієнта, то керувати такою системою складніше. Тут функції додатка розподілені між різними машинами. Необхідність заміни додатка

приводить до його повторної інсталяції на всіх клієнтських комп'ютерах, що вимагає великих витрат, якщо в системі сотні клієнтів.

Поява мови *Java* і аплетів, які завантажуються, дозволили розробляти моделі клієнт/сервер, які знаходяться десь посередині між моделями тонкого і товстого клієнта. Частина програм, що складають додаток, можна завантажувати на клієнтській машині як аплети *Java* і тим самим розвантажити сервер. Інтерфейс користувача будується за допомогою *Web*-броузера, що запускає аплети *Java*.Однак *Web*-броузери від різних виробників і навіть різні версії *Web*-броузерів від одного виробника не завжди виконуються однаково. Більш ранні версії броузерів на старих машинах не завжди можуть запустити аплети *Java*. Отже, такий підхід можна використовувати тільки тоді, коли є впевненість, що у всіх користувачів системи встановлені броузери, сумісні з *Java*.

У дворівневій моделі клієнт/сервер істотною проблемою є розміщення на двох комп'ютерних системах трьох логічних рівнів – представлення, виконання додатка і керування даними. Тому в даній моделі часто виникають проблеми з масштабуванням і продуктивністю, якщо обрана модель тонкого клієнта, або проблеми, пов'язані з керуванням системою, якщо використовується модель товстого клієнта. Щоб уникнути цих проблем, необхідно застосувати альтернативний підхід – трирівневу модель архітектури клієнт/сервер (рис.2). У цій архітектурі рівням представлення, виконання додатку і керування даними відповідають окремі процеси.

Архітектура ПЗ, яка побудована за трирівневою моделлю клієнт/сервер, не вимагає, щоб у мережу були об'єднані три комп'ютерні системи. На одному комп'ютері-сервері можна запустити і виконання додатка, і керування даними як окремі логічні сервери. Водночас, якщо вимоги до системи зростуть, можна буде відносно легко розділити виконання додатка, керування даними і виконувати їх на різних процесорах.

Загалом трирівневу модель клієнт/сервер можна перетворити в багаторівневу, додавши в систему додаткові сервери. Багаторівневі системи можна використовувати і там, де додаткам необхідно мати доступ до інформації, яка знаходиться в різних базах даних. У цьому випадку інтегрувальний сервер (сервер проміжного рівня) розташовується між сервером, на якому виконується додаток, і серверами баз даних (СУБД). Інтегрувальний сервер збирає розподілені дані і представляє їх у додатку подібно до того, як вони знаходяться в одній базі даних. Прикладом трирівневої архітектури може бути система, описана в [3].

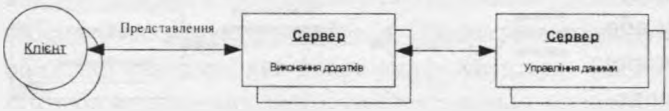

Рис.2. Трирівнева архітектура

Архітектура розподілених об'єктів

У моделі клієнт/сервер розподіленої системи між клієнтами і серверами існують відмінності. Клієнт запитує сервіси тільки в сервера, але не в інших клієнтів; сервери можуть функціонувати як клієнти і запитувати сервіси в інших серверів, але не в клієнтів; клієнти повинні знати про сервіси, які надані визначеними серверами, і про те, як взаємодіють ці сервери.

Така модель добре підходить до багатьох типів програм, але водночас обмежує розробників систем, які змушені вирішувати, де надавати сервіси. Вони також повинні забезпечити підтримку масштабування і розробити засоби включення клієнтів у систему на розподілених серверах.

Більш загальним підходом, який застосовується у проектуванні розподілених систем, є нівелювання відмінностей між клієнтом і сервером і проектування архітектури системи як архітектури розподілених об'єктів. У цій архітектурі основними компонентами системи є об'єкти, які надають набір сервісів через свої інтерфейси. Інші об'єкти викликають ці сервіси, не розріняючи клієнта (користувача сервісу) і сервера (постачальника сервісу).

Об'єкти можуть бути розташовані на різних комп'ютерах в мережі і взаємодіяти за допомогою проміжного ПЗ. За аналогією із системною шиною, яка дозволяє підключати різні пристрої і підтримувати взаємодію між апаратними засобами, проміжне ПЗ можна розглядати як шину програмного забезпечення. Вона надає набір сервісів, які дозволяють об'єктам взаємодіяти між собою, додавати або видаляти їх із системи. Проміжне ПЗ називають брокером запитів до об'єктів. Його завдання – забезпечувати інтерфейс між об'єктами.

Нижче перераховані основні переваги моделі архітектури розподілених об'єктів.

- Розробники системи можуть не поспішати з прийняттям рішень стосовно того, де і як будуть надаватися сервіси.Об'єкти, які надають сервіси, можуть виконуватися в будь-якому місці (вузлі) мережі. Отже, відмінності між моделями товстого і тонкого клієнтів стають несуттєвими, оскільки не потрібно заздалегідь планувати розміщення об'єктів для виконання додатка.
- Системна архітектура є достатньо відкритою, що дозволяє при потребі додавати в систему нові ресурси. Стандарти програмної шини постійно вдосконалюються, що дозволяє об'єктам, написаним на різних мовах програмування, взаємодіяти і надавати сервіси один одному.
- Гнучкість і масштабування. Для того, щоб зменшити системне навантаження, можна створювати екземпляри системи з однаковими сервісами, які будуть надаватися різними об'єктами або різними екземплярами (копіями) об'єктів. При збільшенні навантаження в систему можна додати нові об'єкти, не перериваючи при цьому роботу інших її об'єктів.
- Існує можливість динамічно переконфігурувати систему за допомогою об'єктів, які мігрують у мережі за запитами. Об'єкти, які надають сервіси, можуть мігрувати на той же процесор, що й об'єкти, які запитують сервіси, і цим підвищувати продуктивність системи.

У процесі проектування систем архітектуру розподілених об'єктів можна використовувати двояко:

1. У вигляді логічної моделі, яка дозволяє розробникам структурувати і спланувати систему. У цьому випадку функціональність додатка описується тільки в термінах і комбінаціях сервісів. Далі розробляються способи надання сервісів за допомогою декількох розподілених об'єктів. На цьому рівні, як правило, проектують великомодульні об'єкти, які надають сервіси, що відбивають специфіку конкретної області додатка.

2. Як гнучкий підхід до реалізації систем клієнт/сервер. У цьому випадку логічна модель системи – це модель клієнт/сервер, в якій клієнти і сервери реалізовані як розподілені об'єкти, що взаємодіють за допомогою програмної шини. При такому підході легко замінити систему, наприклад дворівневу на багаторівневу. У цьому випадку ні сервер, ні клієнт не можуть бути реалізовані в одному об'єкті, однак можуть складатися з набору невеликих об'єктів, кожний з яких надає визначений сервіс.

Прикладом системи, якій підходить архітектура розподілених об'єктів, може бути система обробки даних, які зберігаються в різних базах даних (розподілена СУБД). Тут будь-яку базу даних можна представити як об'єкт з інтерфейсом, що надає доступ до даних "тільки читання". Кожний з об'єктів-інтеграторів займається визначеними типами залежностей між даними, збираючи інформацію з баз даних, щоб спробувати відстежити ці залежності.

Об'єкти-візуалізатори взаємодіють з об'єктами-інтеграторами для представлення даних в графічному вигляді або для складання звітів за аналізованими даними.

Для такого типу додатків архітектура розподілених об'єктів підходить більше, ніж архітектура клієнт/сервер, за такими причинами.

1. У цих системах немає одного постачальника сервісу, на якому були б зосереджені всі сервіси керування даними.
2. Можна збільшувати кількість доступних баз даних, не перериваючи роботу системи, оскільки кожна база даних є просто об'єктом. Ці об'єкти підтримують спрощений інтерфейс, який керує доступом до даних. Доступні бази даних можна розмістити на різних машинах.
3. За допомогою додавання нових об'єктів-інтеграторів можна відслідковувати нові типи залежностей між даними.

Головним недоліком архітектур розподілених об'єктів є те, що їх складніше проектувати, ніж системи клієнт/сервер. Виявляється, що системи клієнт/сервер надають більш зрозумілий підхід до створення розподілених систем. Набагато важче розробити систему відповідно до архітектури розподілених об'єктів, оскільки індустрія створення ПЗ поки ще не володіє достатнім досвідом в проектуванні і розробці великомодульних об'єктів.

Постановка задачі

Основним завданням є проектування багаторівневої архітектури для системи забезпечення бізнес-діяльності в масштабах країни. Пропонується п'ятирівнева архітектура інформаційної платформи для розподіленої бізнес-системи з робочою назвою "УніСистема", яка вирішує такі завдання:

- Інформаційні – донесення бізнес-інформації до будь-якого користувача системи.
- Управлінські – вирішення задач керування бізнес-процесом.
- Аналітичні – вирішення задач статистичної обробки та прогнозу бізнес-діяльності.

Метою системи є максимальна автоматизації вирішення описаних задач в середовищі однієї розподіленої системи.

Загальна архітектура системи

Структура системи, наведена на рис.3, передбачає наявність серверів (на першому, другому і третьому рівнях), кожен з яких працює з власно базою даних. Ця структура дозволяє здійснювати описані вище задачі засобами зв'язку з різноманітною пропускною здатністю та їх фізичної організації, а також є оптимальною для розподілсної системи з можливістю масшабування і відповідає адміністративній структурі будь-якої сучасної корпорації класу вище середнього. З огляду на ведення бізнесу корпораціями така п'ятирівнева архітектура з локальною СУБД на кожному рівні забезпечує найвищий рівень захисту від несанкціонованого доступу до СУБД, безпеку збережених даних у ній та найшвидший доступ до даних. Усі програмні компоненти системи зв'язуються між собою, використовуючи мережний протокол *TCP/IP*. Фізичний рівень зв'язку між компонентами не має значення – це може бути локальна мережа, пряме з'єднання до сервера по комутованій або виділеній телефонній лінії, тунельовані Інтернет-канали, корпоративна мережа і т.п. Такий підхід дозволяє створювати розгалужену структуру каналів зв'язку для об'єднання в систему підрозділів корпорації та автоматизованих робочих місць користувачів.

У найпростішому випадку всі компоненти системи – СУБД, серверні служби та автоматизовані робочі місця клієнтів системи (АРМ клієнта) – можуть бути встановлені на одному комп'ютері. У більш складному (типовому) випадку вони можуть бути рознесені по різних комп'ютерах.

Отже, ключові функції системи виконують комунікаційний сервер 1 рівня та прикладний сервер 2 рівня. Саме через них проходять всі запити від АРМ клієнтів у вигляді системних повідомлень та підтримується зв'язок з СУБД. Саме через комунікаційний сервер за допомогою відповідного прикладного сервера різні підрозділи можуть обмінюватися інформацією між собою.

Отже, типова схема організації зв'язку між компонентами системи має такий вигляд (рис.4).

Завдання серверів на кожному рівні є різними. Розглянемо їх детальніше.

- Сервер першого рівня (*S1*, рис.3) є базовим комунікаційним сервером. Основним завданням його є комунікація між головною базою даних та серверами нижнього рівня, а також адміністративні функції. У парі з ним працює глобальна СУБД (*DB0*) системи, яка складає нульовий рівень системи і містить усю інформацію стосовно працівників і бізнес-діяльності корпорації. Керується вміст СУБД логікою, закладеною на сервері і частково засобами самої СУБД. Внесення інформації в глобальну відбувається, в основному, як збирання інформації від серверів першого рівня (*S1_1 ... S1_N*) і є інерційним процесом, який залежить від керованого процесу синхронізації даних. Окрім того, в *DB0* вноситься інформація протоколювання діяльності системи, але лише на рівні власної роботи сервера та комунікації з серверами першого рівня.

  Окремою можливості на *S1* резервується Інтернет-потік.
- За описаними вище розгалуженими каналами зв'язку системи до *S1* під'єднані сервери першого рівня, які можуть називатись регіональними, прикладними чи вузловими серверами (залежно від прикладної області використання системи).

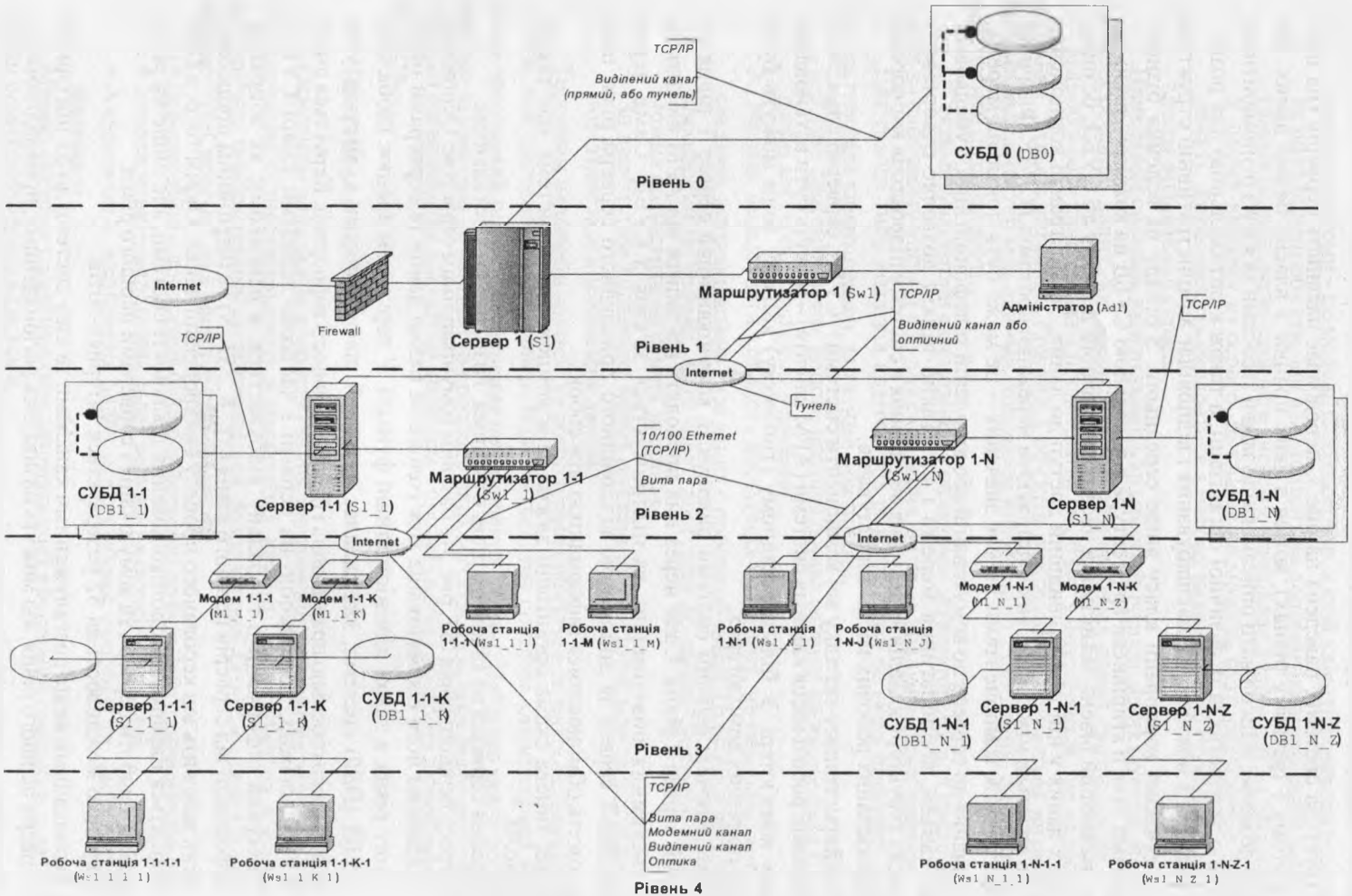

Рис.3. Загальна архітектура систем

Ці сервери є фактичними серверами бізнес-діяльності з власними частковими копіями глобальної СУБД. З кожним прикладним сервером вирішення завдань забезпечує відповідна СУБД другого рівня (*DB 1_1... DB1_N*). СУБД другого рівня є реляційними СУБД, оскільки трафік роботи з бізнес-інформацією є більшим навіть за рівень роботи з СУБД на нульовому рівні.

Сервери *S1_i*, $i = 1..N$ забезпечують безпосередню роботу користувачів системи шляхом надання їм доступу до інформації з СУБД та комп'ютерного формування середовища їх бізнес-діяльності. Уся функціональність, яка надається сервером другого рівня кінцевим користувачам, централізовано розсилається з *S1*.

Сервер першого рівня адмініструється власним адміністратором, який на рис.3 представлений як простий користувач.

- Іншим завданням серверів *S1_i*, $i = 1..N$ є забезпечення діяльності віддалених серверів третього рівня (*S1_1_1... S1_1_K*). Сервери *S1_i_j*, $i = 1..N$, $j = 1..K$ є серверами так званої віддаленої взаємодії, володіють невеликою копією із відповідної СУБД *DB1_i*, $i = 1..N$ і можуть віддалено й автономно функціонувати навіть при тривалій відсутності зв'язку з серверами *S1_i*, $i = 1..N$. Синхронізація вмісту їх баз з DB1_i, $i = 1..N$ відбувається нерегулярно і зрештою може бути здійснена адміністратором другого рівня із зовнішніх носіїв з бізнес-інформацією, які доставлені "вручну". Очевидно, що користувачі серверів *S1_i_j*, $i = 1..N$, $j = 1..K$ фізично можуть не мати можливостей користувачів другого рівня, але вважається, що перших є небагато і працюють вони з дуже малими обсягами інформації.

Принципи адміністрування і контролю доступу

Система "УніСистема" – це мережна система, яка забезпечує багатокористувацький режим роботи з можливістю контролю і обмеження прав доступу.

Функції централізованого контролю доступу покладені на прикладний сервер другого рівня, комунікаційний сервер та СУБД нульового рівня. Як видно з наведеної вище схеми, уся робота користувачів і вся взаємодія компонентів системи між собою реалізовані винятково за такою схемою. Отже, ні користувачі, ні суміжні компоненти системи не мають безпосереднього доступу до баз даних. Такий підхід дозволяє організувати гнучку систему розмежування і контролю доступу до інформації.

Доступ користувачів в систему може відбуватись двояко:

- вручну – через систему паролей і рахунків;
- автоматично – через термінали для зчитування магнітних ідентифікаційних карток користувачів.

Користувач з відповідними правами має доступ в систему з будь-якого сервера (за винятком *S1*), оскільки інформація користувачів накопичується в *DB0* і є доступною інформацією для будь-яких серверів першого рівня. А у випадку існування зв'язку є доступною і для серверів другого рівня.

Весь мережний обмін інформацією шифрується з застосуванням сучасних стійких до дешифрування алгоритмів і в такій формі зберігається в СУБД на усіх рівнях. Ключі шифрування також динамічно міняються в кожнім сеансі з застосуванням су-

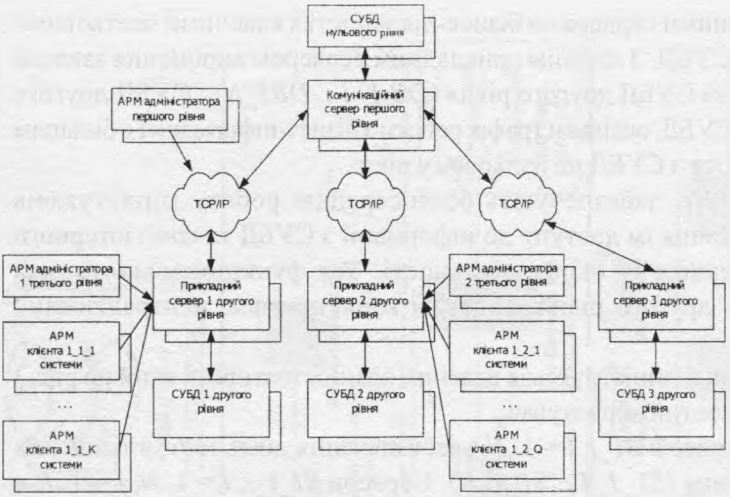

Рис.4. Організації зв'язку між компонентами системи

часних стійких алгоритмів договору про ключі. Такій систематичній зміні коду шифрування підлягають лише паролі, які забезпечують комунікації на рівні користувачів і доступних для них серверів. Внутрішні комунікації здійснюються за допомого інших алгоритмів і їх ключі є недоступними для зовнішніх користувачів. Отже, система захищена від прослухо-

вування каналу зв'язку, і жодна інформація не з'являється в каналі зв'язку у відкритому вигляді. Це дозволяє використовувати для побудови структури всілякі, у тому числі і глобальні, мережі (наприклад, Інтернет).

Структура інформаційних потоків

Структурну схему інформаційних потоків в системі наведено на рис.5. Інформаційна платформа забезпечує передачу бізнес-інформації між рівнями для накопичення інформації, повноцінного функціонування модулів системи, збереження даних і т.д. відповідно до *функціонального набору модулів та по горизонталі для зв'язку з фінансовими* модулями, котрі можуть бути винесені з системи. У цьому випадку зв'язок з фінансовим контуром на кожному рівні відбувається за допомогою тісної інтеграції даних в середовищі збереження даних. Це зокрема дозволяє, наприклад, результати проведеної операції в одному з модулів практично в один момент відобразити у відповідних рахунках бухгалтерського обліку.

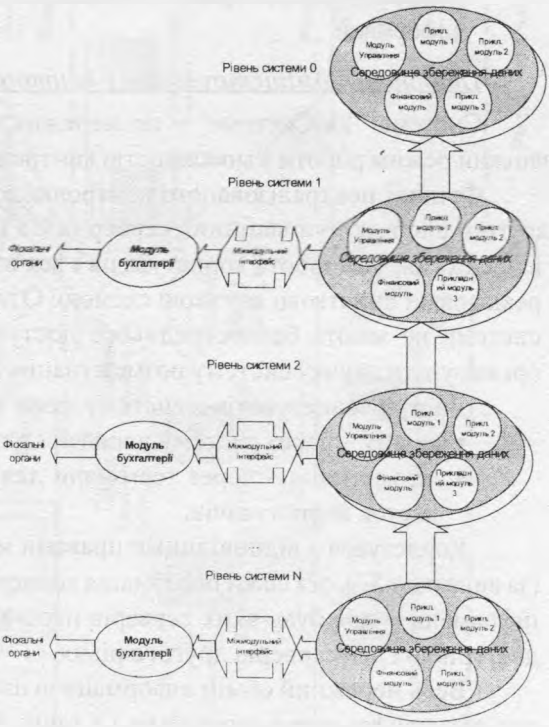

Рис.5. Структурна схема інформаційних потоків в системі

Передача інформації базується на системі обміну повідомленнями. Кожне повідомлення містить інформаційну та службову частини. Службова частина

складається з ідентифікатора типу повідомлення, ідентифікатора повідомлення, адрес відправника та місця призначення, набору прапорців із службовою інформацією. Службова та інформаційна частина об'єднуються в один неперервний блок, що дозволяє передавати повідомлення по *TCP/IP* без додаткових перетворень.

На кожному рівні, окрім найвищого, існує можливість аналітичної обробки даних та видавання результатів цієї обробки назовні. Кожен із рівнів містить модуль управління, який керує його роботою, та набір прикладних модулів, які виконують конкретні задачі.

З огляду на структуру інформаційних потоків головними рисами її є:

– збереження принципу ієрархічності;
– агрегація даних від рівня до рівня;
– нарощення функціональної потужності з рівня до рівня та забезпечення планування та аналізу результатів діяльності компанії.

Архітектура сервера першого рівня

Загальна схема сервера *S1* наведена на рис.6. Як видно з рисунка, сервер є багатопотоковою мережною системою, яка виконує завдання підтримки комунікації віддалених користувачів з *DB0*, роботу з адміністрування, моніторинг системи та ін.

Побудований базовий сервер за трирівневою архітектурою. Вона передбачає:

- 0-й рівень – рівень ядра сервера.
- 1-й рівень – прикладний рівень.
- 2-й рівень – рівень зовнішніх відносно сервера користувачів.

Розглянемо кожен з рівнів детальніше.

Рівень ядра

На нульовому рівні функціонує ядро сервера. Його основними завданнями є управління прикладними потоками, маршрутизація повідомлень та моніторинг роботи системи. Окрім того, ядро керує основною чергою повідомлень системи. Зокрема саме функція ядра аналізує пріорітетність та вибирає повідомлення з основної черги. Прикладні потоки стосовно основної черги можуть лише генерувати повідомлення і додавати їх в основну чергу.

З метою пришвидшення роботи з чергою, особливо у випадку появи в системі повідомлень з вищим пріорітетом, основна черга складається з двох черг

$$Q^{Thread\,0} = \begin{cases} Q_{\text{пр}}^{Thread\,0} & - \text{ пріоритетна підчерга;} \\ Q_{\text{зв}}^{Thread\,0} & - \text{ звичайна підчерга;} \end{cases}$$

тут $Q^{Thread\,0}$ – черга. Верхній індекс вказує рівень, на якому керується черга.

Першою завжди проглядається $Q_{\text{пр}}^{Thread\,0}$. Лишу у випадку, коли $Q_{\text{пр}}^{Thread\,0}$ є порожньою, переглядається підчерга $Q_{\text{зв}}^{Thread\,0}$. Така схема дозволяє уникнути зайвих пошуків пріорітетних повідомлень у випадку існування єдиної черги.

Функція маршрутизації та керування $Q^{Thread\,0}$ є таймерною. Це дає можливість розвантажувати ядро і передавати управління іншим функціям. Значення таймера є динамічною величиною, а це дає можливість керувати роботою таймера рівня ядра базового сервера (основного таймера).

Рівень прикладних потоків

На першому рівні фунціонують прикладні потоки. Залежно від призначення і функціонування вони можуть бути буферизованими – типу *A*, простими (небуферизованими) – типу *B* або з власною локальною чергою (потоковою чергою) – типу *C*.

До типу *A* відносять: потік адміністратора, потік синхронізації та інтернет-потік.

До типу *B* відносять функціональний потік.

До типу *C* відносять потік до бази даних та мережний потік.

Розмір буфера в потоках типу А є керованим, що дає можливість або динамічно визначати необхідний розмір буфера залежно від завантаженості потоку, або керувати його розміром "вручну".

Черги в потоках типу *C* є чергами, які побудовані подібно до основної черги. Це означає, що вони містять такі ж самі об'єкти і мають вигляд

$$Q^{Thread\ j} = \begin{cases} Q_{\text{пр}}^{Thread\ j} & - \text{пріоритетна підчерга;} \\ Q_{\text{зв}}^{Thread\ j} & - \text{звичайна підчерга;} \end{cases}$$

тут $j = 1, 2$. При $j = 1$ отримуємо чергу мережного потоку, а при $j = 2$ – чергу потоку до БД.

Керуються потокові черги таймерними функціями, але вже потоків. Відповідно до цього усе управління потоковими чергами покладено на самі потоки.

Усі черги працють в асинхронному режимі. Це досягається тим, що черги рознесені по різних областях використання і керуються окремими таймерними функціями.

Прості потоки також отримують повідомлення від ядра. Оскільки вони є небуферизованими і не мають власної черги, то єдиним способом передавання їм інформації є використання засобів синхронізації операційної системи.

Зовнішній рівень

На другому рівні працють зовнішні відносно базового сервера користувачі. Серед них є вузлові сервери, інтернет-клієнти, база даних і адміністратор. Архітектури програмного забезпечення для кожного з наведених користувачів не надається.

Прикладні потоки сервера нульового рівня

Розглянемо детальніше основні потоки базового сервера (рис.6)

Мережний потік (Thread_1)

Цільове призначення мережного потоку є забезпечення з'єднання зі серверами *S1_i*, $i = 1..N$. У випадку отримання запиту на нове з'єдання в основну чергу передається команда на створення нового потоку. Додатково в новостворений потік передається уся необхідна інформація про нового клієнта. Надалі уся комунікація між *S1* та *S1_i*, $i = 1..N$ здійснюється через новостворений потік. Детальніше механізм з'єднання описано нижче.

Запропонована схема робота буде задіяна лише в тому випадку, коли існує пряме з'єднання з серверами *S1_i*, $i = 1..N$. У випадку використання мехнанізму тунелю дочірні потоки не забезпечують комунікації з серверами *S1_i*, $i = 1..N$. Це завдання покладається саме на *Thread 1*. Зумовлене це особливостями організації зв'язку через Інтернет.

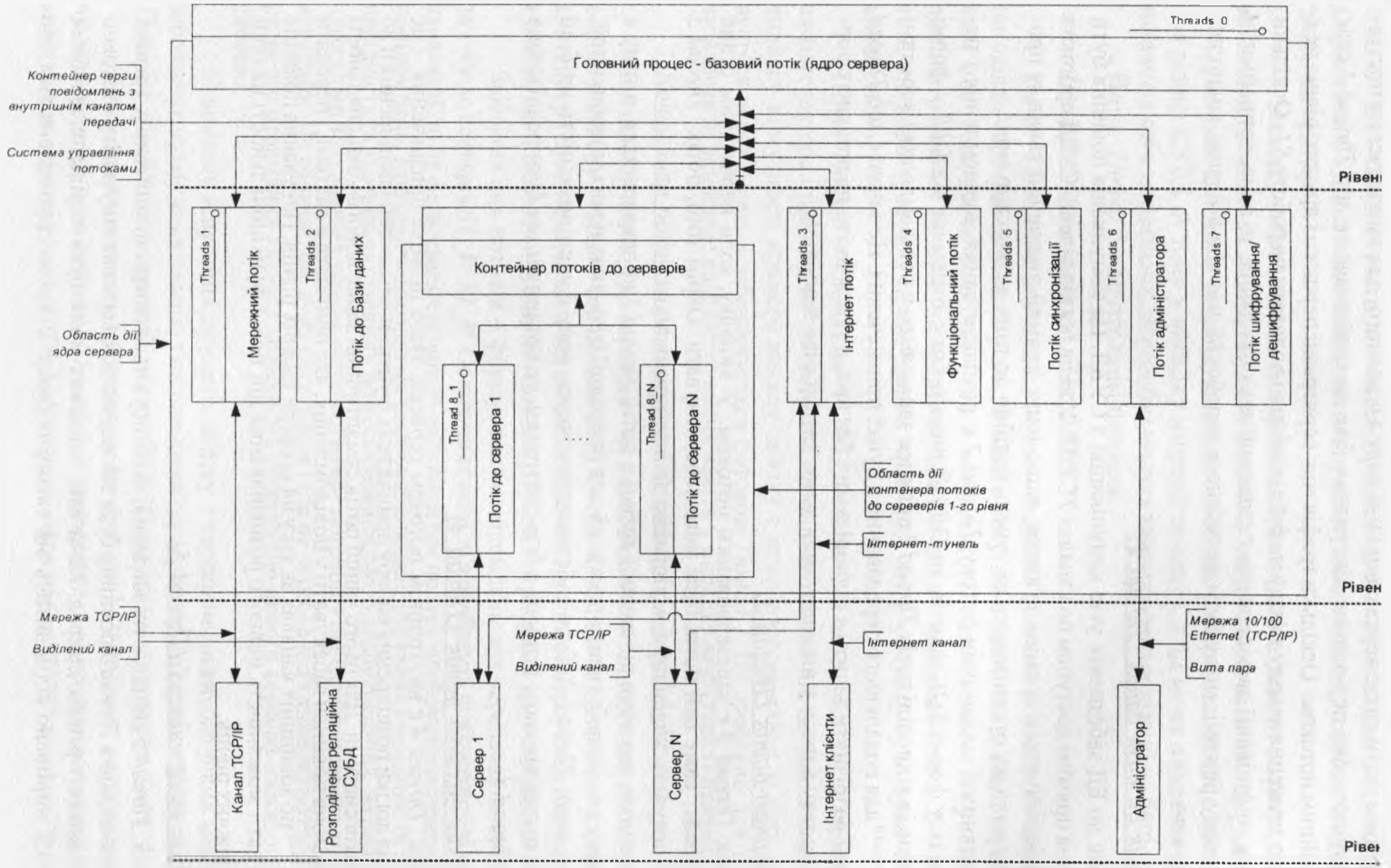

Рис.6. Архітектура базового сервера

У випадку тунелювання сервер *S1* стає фактично *Web*-сервером і повинен увесь зв'язок підтримувати лише через один 80-й порт. Окрім того, для забезпечення постійного зв'язку потрібне періодичне пінгування. Усе це спричиняє те, що *Thread 1* стає комунікаційним потоком. Оскільки трафік між серверами першого і другого рівнів не є великим, то звуження каналу зв'язку не буде особливо впливати на роботу *S1* . Оскільки основним у комунікації двох серверів є синхронізація баз даних, то вона, принаймні в автоматичному режимі, може бути винесена в неробочий час або в період, коли система є вільною.

Потік до бази даних (Thread 2)

Потік до БД забезпечує усю комунікацію з СУБД. Ця комункація повинна бути зреалізована прямим доступом по каналах *TCP/IP*. У випадку віддаленої СУБД протокол *TCP/IP* забезпечується прямим зв'язком, якщо існує прямий виділений канал, у протилежному випадку організовується тунель подібно до того, як описано вище.

Основними завданнями потоку *Thread 2* є розпаковування повідомлення і передавання їх у формі *SQL*-запитів до *DB0*. З іншого боку, отримані від *DB0* у формі наборів записів відповіді потік *Thread 2* повинен запакувати у повідомлення і передати його у $Q^{Thread0}$ для подальшого передавання в пункт призначення. Це означає, що *Thread 2* змушений розрізняти запити з відповідями і без них, а також вміти правильно сформувати зворотну адресу у випадку, коли запит передбачає відповідь.

Інтернет-потік (Thread 3)

Потік *Thread 3* є зарезервованим потоком. У випадку, коли необхідним стане *Web*-інтерфейс, саме цей потік буде його забезпечувати. Окрім того, потік *Thread 3* буде забезпечувати класичну комунікацію за *http*-протоколом.

Зазначимо, що поки що жодної логіки в потік *Thread 3* не закладається, а тому в майбутньому він може бути використаний для реалізації будь-яких нових можливостей. У випадку, коли *Thread 3* буде забезпечувати якусь нову функціональність на боці *Sw1*, а при цьому виникне потреба в *http*-підтримці, то остання може бути реалізована в потоці *Thread 1*.

Функціональний потік (Thread 4)

Потік *Thread 4* є внутрішнім потоком сервера. Його цільове призначення – це розгортання логіки розширення набору прикладних функціональностей і поширення її до рівня клієнтів системи. Для цього даний потік семантичний запис нової функціональності з *Plug-In* файлів повинен запакувати у повідомлення, яке прийняте в системі, і передати через $Q^{Thread0}$ по зовнішніх каналах зв'язку на сервери нижніх рівнів. Пакування повинне забезпечувати можливість повного розпізнавання дій нової функціональності на боці серверів нижчих рівнів.

Потік синхронізації (Thread 5)

Потік *Thread 5* вирішує усі завдання, які стосуються синхронізації даних з різних БД. Це означає, що в *Thread 5* повинно бути закладено усю математичну логіку стосовно інтеграції даних з різних джерел. Хоча таке завдання може бути вирішено засобами самої СУБД, вирішено відмовитись від такого підходу. Відмова спричинена тим, що

ускладнюється потік *Thread 2,* оскільки саме він повинен викликати синхронізацію на боці СУБД і забезпечувати для цього процесу дані (що не є тривіальною задачею). Окрім того, СУБД не може забезпечити гнучкості синхронізації, тобто не існує способів зміни алгоритму.

У випадку забезпечення синхронізації засобами власного потоку з'являється можливость зміни як алгоритму, так і режиму самої синхронізації. Нарешті, кожна СУБД володіє власними індивідуальними засобами синхронізації, які є несумісними для різних СУБД. А тому у випадку міграції на нову СУБД може з'явитися проблема несумісності і, як наслідок, перероблка модуля виклику синхронізації на боці СУБД.

Потік *Thread 5* є внутрішнім потоком.

Потік адміністратора (Thread 6)

Потік *Thread 6* є звичайним користувацьким потоком подібно до аналогічних потоків на серверах нижнього рівня. З іншого боку, він є фактично єдиним користувацьким потоком. Хоча технологічно є можливість одночасного існування двох адміністративних потоків, на практиці така ситуація є малоімовірною. Зумовлено це тим, що користувачів з таким рівнем прав доступу в систему буде дуже обмежена кількість.

Потік *Thread 6* каналами *TCP/IP* забезпечує комунікацію з комп'ютером адміністратора і реалізацію допустимих адміністративних дій на боці сервера *S1.* У випадку відсутності прямого каналу зв'язку з комп'ютером, канал зв'язку організовується через тунель і забезпечується потоком *Thread 1.* Це зроблено для того, щоб забезпечити віддалене адміністрування навіть у випадку відсутності прямого зв'язку.

Контейнер потоків до серверів (Thread 1_1.. Thread 1_N)

Контейнер потоків до серверів – це є класичний контейнер, який побудований на основі шаблону <*vector*> бібліотеки *STL.* Це дає можливість динамічно наповнювати контейнер і бути незалежним від кількості елементів контейнера. Вмістом контейнера є класи, які описують похідні потоки для роботи з серверами другого рівня.

Завдання цих потоків – формування повідомлень для передавання їх каналами зв'язку до серверів *S1_1.. S1_N.* Оскільки сеанси зв'язку будуть відбуватись відкритими каналами, то це формування повинне включати процедуру шифрування/дешифрування повідомлень. Критерієм алгоритму шифрування/дешифрування повинна бути висока швидкість та задовільна стійкість до несанкціонованого дешифрування. За основу взято алгоритми з закритим ключем, оскільки немає потреби у передаванні ключа відкритими каналами. Це означає, що нова реєстрація (але не чергове під'єднання) потребує попереднього отримання ключа і дозволу на реєстрацію нового сервера нижнього рівня в системі.

У випадку існування прямого зв'язку з сервером нижнього рівня кожен потік забезпечує пряму комунікацію з відповідним сервером другого рівня. Наприклад, потік *Thread 1_1* забезпечує прямий зв'язок *TCP/IP* каналами з сервером *Sw 1_1.*

Якщо прямого зв'язку з серверами *Sw 1_1*, $j = 1, N$ не існує, то зв'язок забезпечується через тунель потоком *Thread 1.* У цьому випадку усі повідомлення, які призначені для серверів *Sw 1_1*, $j = 1, N$ передаються через $Q^{Thread\,0}$ в потік *Thread 1* для

подальшої передачі їх назовні. З іншого боку, усі зовнішні повідомлення від серверів *Sw 1_1*, *j* = 1, *N*, які приходять в зашифрованому вигляді потоком *Thread 1* через $Q^{Thread\ 0}$, передаються в потоки *Thread 1_j*, *j* = 1, *N* для їх дешифрування і подальшого виконання.

Висновки

Розробники архітектур клієнт/сервер, обираючи найбільш придатну архітектуру, повинні враховувати ряд факторів. Зокрема, дворівневу архітектуру тонкого клієнта треба використовувати у випадку побудови таких програм:

- Наслідувані системи, в яких недоцільно розділяти виконання програми і керування даними.
- Програми з інтенсивними обчисленнями, наприклад компілятори, але з незначним обсягом керування даними.
- Програми, в яких обробляються великі масиви даних (запити), але з невеликим обсягом обчислень в самій програмі.

Дворівнева архітектура товстого клієнта використовується при побудові таких програм:

- Програми, де користувачу потрібна інтенсивна обробка даних (наприклад, візуалізація даних або великі обсяги обчислень).
- Програми з відносно постійним набором функцій на боці користувача, що застосовуються у середовищі з добре налагодженим системним керуванням.

І, нарешті, трирівневу і багаторівневу архітектуру клієнт/сервер треба використовувати у випадку розробки таких програм:

- Великі програми із сотнями і тисячами клієнтів, об'єднаних засобами зв'язку з різноманітною пропускною здатністю та різної фізичної організації.
- Програми, в яких часто змінюються і дані, і методи клієнт/сервер-обробки.
- Інформаційні розподілені системи, в яких виконується інтеграція даних з багатьох джерел, як правило, територіально рознесених.

Запропонована в роботі архітектура системи є багаторівневою, а тому її основною сферою використання є побудова великих бізнес-систем. У випадку потреб невеликої інформаційної системи недоречно використовувати таку громіздку систему з закладеною в неї надлишковістю підтримки теоретично будь-яких бізнес-функціональностей. З іншого боку, запропонована система може охоплювати величезні регіони масштабу країни, а її функціональність обмежується лише можливостями СУБД.

Щоби запобігти залежності від СУБД, уся логіка роботи з даними в системі винесена на рівневі сервери. Це також дало можливість отримати достатньо легку міграцію з однієї СУБД на СУБД іншого вищого класу.

Ще однією перевагою пропонованої системи є абсолютна мобільність усіх зареєстрованих користувачів системи. Шлюз доступу в систему для будь-якого клієнта існує на усіх серверах третього рівня.

І, нарешті, зазначимо, що система є універсальною. Як видно з архітектури базового сервера, усе бізнес-функціональне наповнення є *Plug-Ins* модулями. *Plug-Ins* модулі містять не лише прикладні функції, а й користувацький інтерфейс. Це робить систему придатною до використання практично в усіх галузях бізнес-діяльності людини.

1. І. Соммервилл. Инженерия программного обеспечения: Пер. с англ.– М.: Издательский дом "Вильямс", 2002. – 624с.
2. Coulouris G. Dollimore J. et al. Distributed Systems: Concepts and Design. – Wokinham: Addison-Wesley. 1994.
3. Рашкевич Ю., Пелешко Д., Пасєка М., Стецюк А. Проектування Web-орієнтованих розподілених навчальних систем. Conference Telematics and Life-Long Learning. Proceedings of the International WorkShop TLLL-2001, October 15-17, 2001, Kyiv, Ukraine. Pp. 143–152.
4. Калужний Б., Пелешко Д., Маляр А., Яремко Р. Проектування системи автоматичного контролю і керування глибиннопомповим видобутком нафти. Технічні вісті, 2002/1(14), 2(15). – С. 87–95.
5. Чабан В.Й., Пелешко Д.Д. Методологія побудови інформаційних систем для туристичної галузі. Праці Другого міжнародного конгресу "Інформатизація рекреаційної та туристичної діяльності: перспективи культурного та економічного розвитку", Трускавець 6-9 жовтня (ДНДІІ). – С. 40–47.
6. Francisco J. Ballesteros and Luis L. Fern'andez. The Network Hardware is the Operating System. In Proceedings of the 6th Hot Topics on Operating Systems (HotOS-VI)., Cape Cod, MA (USA), 1997.

Л.Лукащук, Н.Кустра
Національний університет "Львівська політехніка"

УДК 681.142.6

ВИКОРИСТАННЯ МУЛЬТИПЛЕКСОРІВ ДЛЯ ВІДТВОРЕННЯ ЛОГІЧНИХ ФУНКЦІЙ

Розглянуто способи відтворення логічних функцій за допомогою мультиплексорів, аналізується ефект перегонів.

In this article it is considered ways of display of logic functions with the help of multiplexers the effect of races is analyzed.

Вступ

Логічні схеми на основі мультиплексорів належать до пристроїв,які налагоджуються на вирішення тієї чи іншої задачі. Універсальність цих схем полягає у