УДК 004.032.2

В. І. Яркун, Я. С. Парамуд
Національний університет “Львівська політехніка”,
кафедра електронних обчислювальних машин

АЛГОРИТМІЧНО-ПРОГРАМНІ ЗАСОБИ СИНХРОНІЗАЦІЇ ПРИ ОБМІНІ ДАНИМИ ВЕЛИКИХ ОБСЯГІВ

Проаналізовано використання керуючих засобів щодо конфігурації потоків для ефективного передавання даних великих обсягів та наведено основні переваги їх використання. Розкрито задачу синхронізації декількох процесів та її розв’язання за допомогою задачі обмеженого буфера. При надсиланні повідомлення великого обсягу рекомендовано його розбиття на певну кількість процесів, за синхронну роботу яких відповідають запропоновані засоби, універсальні для різних операційних систем.

Ключові слова: проблема синхронізації декількох процесів, задача обмеженого буфера, структура постачальник-споживач.

ALGORITHMIC AND SOFTWARE SYNCHRONIZATION OF INFORMATION EXCHANGE

In this article is considered the use of tools for configuration managing flow for efficient transferring large amount of data and described the main benefits of using them. Solved the problem of synchronization of multiple processes by producer-consumer design. When sending large amount of data it is recommended to divide it into several small parts for proper work of which are responsible the proposed configurations.

Key words: producer-consumer pattern, bounded-buffer problem, multi-process synchronization problem.

Вступ

Стрімкий розвиток інформаційних технологій вимагає в багатьох випадках обробки, передання, використання великих обсягів даних. Актуальним стає дефіцит часу, який потрібно затрачати на синхронне даних керування потоками, на перевірку стосовно їх правильності та цілісності. При обміні даними великих обсягів із використанням мережі Інтернет може бути неефективним використання ресурсів користувача. Отже, задача синхронізації обміну великими обсягами даних із використанням мережі Інтернет є актуальною.

Окреслення проблеми

Наявність декількох потоків в програмі може спричинити виникнення проблем, що стосуються безпечного доступу до ресурсів від різних потоків. Два потоки, які спричиняють зміни в одному ресурсі, можуть заважати один одному в непередбачений спосіб. Наприклад: один потік може перезаписати зміни, спричинені іншим потоком, що, своєю чергою, може зіпсувати певні налаштування або спричинити певний невідомий стан аплікації під час роботи з ресурсами. Коли йдеться про безпеку потоку, вдалий дизайн є одним із найкращих захистів. Для запобігання небажаній зміні даних потоками доцільно застосувати такий алгоритм, щоб мінімізувати затрати користувача на засоби синхронізації, забезпечивши їх максимальну ефективність. Незважаючи на те, що мінімізація впливу засобів синхронізації є привабливішою, вона не завжди можлива. Переважно потрібно використовувати компромісні рішення.

Аналіз останніх досліджень та публікацій

Завдання синхронізації процесів та використання спільного буфера не є новим у комп'ютерних технологіях. Розв'язанню цього завдання присвячено дослідження багатьох вчених, серед яких можна виділити роботи Т. Бройнля та В. В. Корнєєва [1, 2]. Запропоновано загальні підходи із використанням семафорів, моніторів та інших засобів синхронізації. Оскільки в переважній більшості рішення є компромісними, пошук ефективних рішень для конкретного застосування є актуальним.

Сьогодні провідні комп'ютерні фірми пропонують певні технології для синхронізації процесів роботи в програмах. Слід взяти до уваги такі програмні засоби для керування потоками в мові програмування Objective-C: Operation objects – це обгортки для задач, які будуть виконуватись на іншому потоці; Grand Central Dispatch(GCD) – ще одна альтернатива керуючих засобів потоків, що дасть змогу скоцентруватися на задачах, які потрібно виконати, а не на управлінні потоками; Idle-time notifications – засіб керування невеликою кількістю потоків для задач, які є відносно короткими і мають низький пріоритет; Asynchronous functions – системні інтерфейси, що містять багато асинхронних методів, які забезпечують автоматичний паралелізм; Timers – засіб керування одним потоком; можна використовувати таймери в програмі на головному потоці для періодичних задач, які є занадто тривіальними для того, щоб надавати їм окремий потік [3, 4].

Але ці методи прив'язані до певного середовища програмування, певної мови програмування, операційної системи, що, своєю чергою, змусить розробника затрачати час для створення різного дизайну та структури синхронізації в програмному забезпеченні для відповідних мов програмування під різні платформи [3, 5, 6].

Мета статті

Метою цієї роботи є створення алгоритмічно-програмних засобів для комп'ютерних пристроїв, які повинні отримувати інформацію від глобальної мережі Інтернет та забезпечити її передавання ресурсам системи споживача без втрат та бути універсальними під різні найпоширеніші операційні системи.

1. Основний матеріал дослідження

Вирішування завдання ефективного обміну великими обсягами даних із використанням мережі Інтернет доцільно на основі алгоритму до задачі обмеженого буфера, який описує два потоки роботи: перший – це потік постачальника (надалі використовуємо слово "постачальник"), другий – потік споживач (надалі використовуємо слово "споживач"), які використовують спільний буфер наперед встановленого розміру [7]. Задачею постачальника є утворення фрагментів даних та передавання їх до буферу – це циклічний процес, який виконують певну кількість разів, одночасно з цим споживач забирає дані з буфера, цим самим його і спорожнює для подальшого використання даних. Отже, завданням є не допустити запису постачальником в повний буфер і не дозволити споживачу забирати дані з пустого буфера.

Для різних вимог використання комбінація кількості постачальників та споживачів може бути різною – це залежить від постановки задачі та результату, якого потрібно досягти. Можливі такі варіанти комбінацій:

1) один постачальник – один споживач (цей варіант розглянуто вище)
2) один постачальник – декілька споживачів
3) декілька постачальників – один споживач
4) декілька постачальників – декілька споживачів

Розглянемо алгоритм роботи передавання даних з одним постачальником та одним споживачем. Для функціонування цього алгоритму слід створити два окремі класи, які б забезпечували роботу як окремі два потоки, що працюють по черзі. В кожному класі буде свій власний метод (процедура), який власне і виконує функцію для свого потоку.

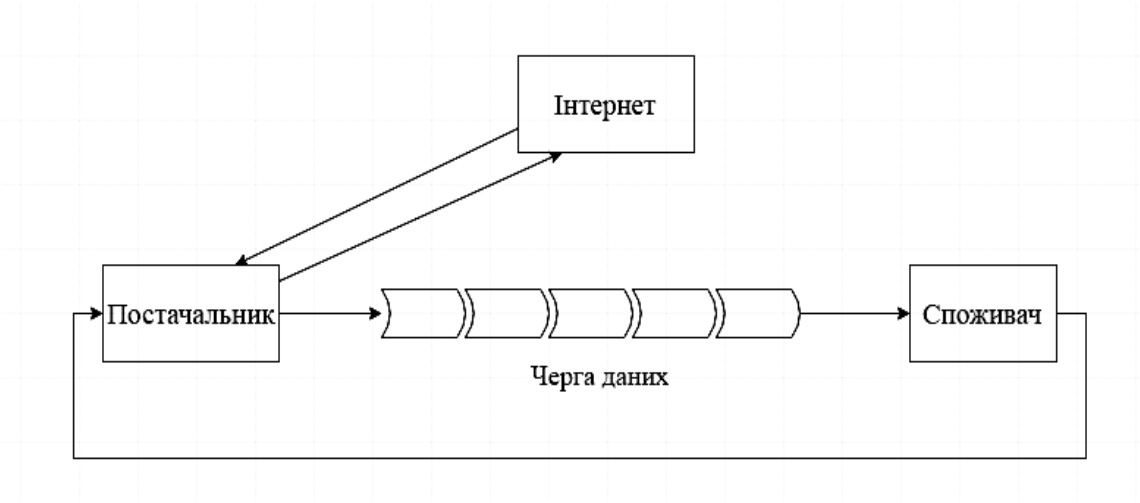


Рис. 1. Узагальнена схема роботи постачальника-споживача

На рис. 1 зображено абстракцію цієї задачі, де постачальник забезпечує даними проміжний буфер, звідки споживач буде їх забирати. Постачальник отримує дані, в цьому випадку з Інтернету. Інтернет використано як джерело даних, але він може отримувати їх з будь-якого іншого зовнішнього ресурсу, наприклад, із сервера. Постачальнику необхідно робити до Інтернету запити аби отримати дані, а споживачу, своєю чергою, потрібно повідомляти постачальника про початок роботи. У цьому прикладі використано шаблон з одним постачальником та одним споживачем, хоча їх може бути більше.

Роботу вищенаведеного дизайну забезпечують свого роду затримки. Ці затримки будуть ставити споживача в режим очікування, поки постачальник записує дані в буфер, і навпаки – постачальник чекатиме, поки не спорожниться буфер. Використання затримок зумовлює застосування так званих семафорів, які блокують потік до подальшого використання. Вищезгадані семафори є “замком” для цього потоку. Розрізняють такі види замків: mutex, recursive lock, read-write lock, distributed lock, spin lock double-checked lock, condition lock[4]. Можна використовувати окремі з цих замків або їхню комбінацію. Але потрібно бути обережним, аби не спричинити deadlock (обидва потоки в режимі wait, очікують на роботу іншого) та livelock (неправильне використання в одному потоці двох або більше замків). Слід правильно розташовувати замки, щоб не витрачати на їх блокування та розблокування багато ресурсів – саме це основне для цієї задачі.

Також для того, щоб два потоки могли “спілкуватися”, їм потрібно надати доступ до певного спільного ресурсу, який вони би використовували. В цьому прикладі буде використано масив як вмістилище даних, власне масив і міститиме файли, які постачальник завантажуватиме з Інтернету, а споживач братиме їх із масиву. Масив у цьому випадку – це абстракція для контейнера, що містить певні дані, які використовує система, тому як для опису алгоритму надалі буде використано слово “буфер”.

На рис. 2 зображено схему алгоритму роботи постачальника, який робить запит до зовнішнього джерела даних – Інтернету, після чого завантажує їх із зовнішнього ресурсу та записує дані до буфера. Постачальник постійно завантажує дані і записує їх до буфера. В кінці циклу перевіряють, чи буфер повністю заповнено. Якщо ні, то процес повторюється заново. Якщо постачальник завершив роботу, про що сигналізує буфер, коли він повністю заповнений, то постачальник звільняє доступ до нього. При цьому надається можливість доступитися іншим об’єктам до нього, зокрема і споживачу. Споживач, своєю чергою, отримує дані з буфера, після цього його очищає. Цей процес є циклічним завдяки перевірці стану буфера. Завершуючи роботу, споживач теж відкриває доступ до буфера іншим ресурсам, зокрема постачальнику.

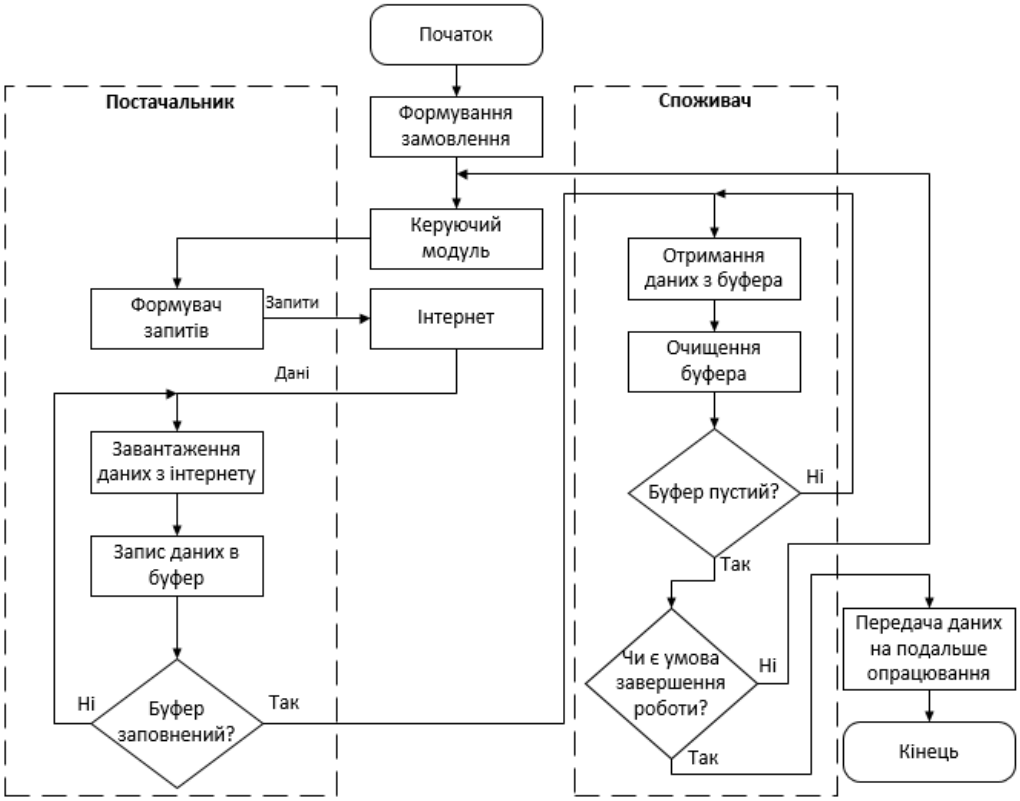


Рис. 2. Загальна схема алгоритму функціонування системи в однозадачному режимі

Отже, цей алгоритм показує універсальну для різних мов програмування роботу системи під час передавання даних. Цей алгоритм забезпечує роботу одного постачальника та одного споживача. Використовуючи задачу обмеженого буфера для декількох систем, мультикористувацьких запитів чи під час передавання великих обсягів даних, слід розширити цей алгоритм з використанням декількох постачальників та декількох споживачів.

Один із варіантів програмної реалізації розглянутого алгоритму має такі особливості. Для застосування вищенаведеного алгоритму використано мову високорівневого програмування – Objective C [7].

Нижче наведено приклад програми класу постачальника:

```
-(void)addObject: (id) object
{
    pthread_mutex_lock(&lock);
    Node * tempNode = [[Node alloc] init];
    tempNode.data = object;

    if (self.last)
    {
        self.last.next = tempNode;
        self.last = tempNode;
    }
    if (self.first == nil)
    {
        self.first = self.last = tempNode;
    }

    pthread_cond_signal(&cl);
    pthread_mutex_unlock(&lock);
}
```

Підпрограма 1

У підпрограмі 1 реалізовано основний метод класу постачальника, який, власне, і виконує головну роботу. У вищенаведеному коді використано технологію POSIX Thread, які переважно використовують для UNIX систем, оскільки вона є доволі зручною та універсальною. Отже, спочатку виконується блокування потоком доступу до буфера, потім перевірка для запису в кінець масиву, після чого перевіряється, чи існує початок запису. Оскільки від самого початку цей метод заблоковує буфер, після всіх перевірок він подає сигнал, що завершив роботу, і розблоковує буфер для подальшого його вжитку іншими ресурсами.

Нижче наведено фрагмент програми, який демонструє роботу класу споживача:

```
-(id)getObject
{
    pthread_mutex_lock(&lock);

    while(self.first == nil)
    {
        pthread_cond_wait(&cl, &lock);
    }

    id data = self.first.data;

    self.first = self.first.next;


    pthread_mutex_unlock(&lock);
    return data;
}
```

Підпрограма 2

Підпрограма 2 дає можливість побачити основний метод, який повинен бути імплементований у клас споживача. Спочатку метод заблоковує доступ до ресурсів, після чого йде перевірка на термін блокування – тобто, допоки не прийметься все вмістиме буфера аж до самого початку (початок – це self.first), доти буде заблоковано доступ до ресурсів. Після цього код забирає дані з буфера, розблоковується доступ до закритих ресурсів, і повертаються дані. Аналогічно за наведеними підходами пишеться код програми іншими мовами програмування(C#, Java, Swift та інші) для різних операційних систем та пристроїв.

Одним із застосувань є мультизадачний режим функціонування комп'ютерного пристрою. Як вже було згадано, можливо, що декілька постачальників та споживачів паралельно працюватимуть, що забезпечить швидкодію та вирішить завдання обміну великими обсягами даних. Варіанти "один постачальник – декілька споживачів" та "декілька постачальників – один споживач" є похідними від попередніх, тому основну увагу звертають на розгляд варіанта із декількох постачальників та декількох споживачів.

Застосування такого варіанта є доволі складним. Реалізуючи цей варіант, слід синхронізувати порядок запису постачальником даних до буфера та їх читання. Ця проблема суттєво ускладнюється з використанням спільного буфера для декількох об'єктів (більше двох). При використанні спільного буфера для декількох постачальників та споживачів слід взяти до уваги їх роботу щодо запису в чергу даних та отримання даних з черги. Якщо цим нехтувати, то постачальник, наприклад, може перезаписати дані іншого постачальника, а споживач, своєю чергою, може отримати дані з черги, які вже отримав інший споживач. Для коректної синхронізації декількох об'єктів, що використовують спільну чергу даних, слід ставити додаткові замки на кожен об'єкт, а також зважати на послідовність їх роботи. Наведений вище алгоритм можна застосовувати і для цього випадку, тільки необхідно забезпечити попередньо згадані умови, а також буфер має бути захищеним.

Раніше згаданий алгоритм можна застосовувати для передавання даних різними потоками, кожен з яких використовуватиме одного постачальника – одного споживача; кожна пара постачальника-споживача має свій окремий буфер (чергу), якою керують тільки вони, не заважаючи іншим у роботі. Запропоновану схему алгоритму роботи декількох постачальників і декількох споживачів із окремими чергами даних наведено на рис. 3.

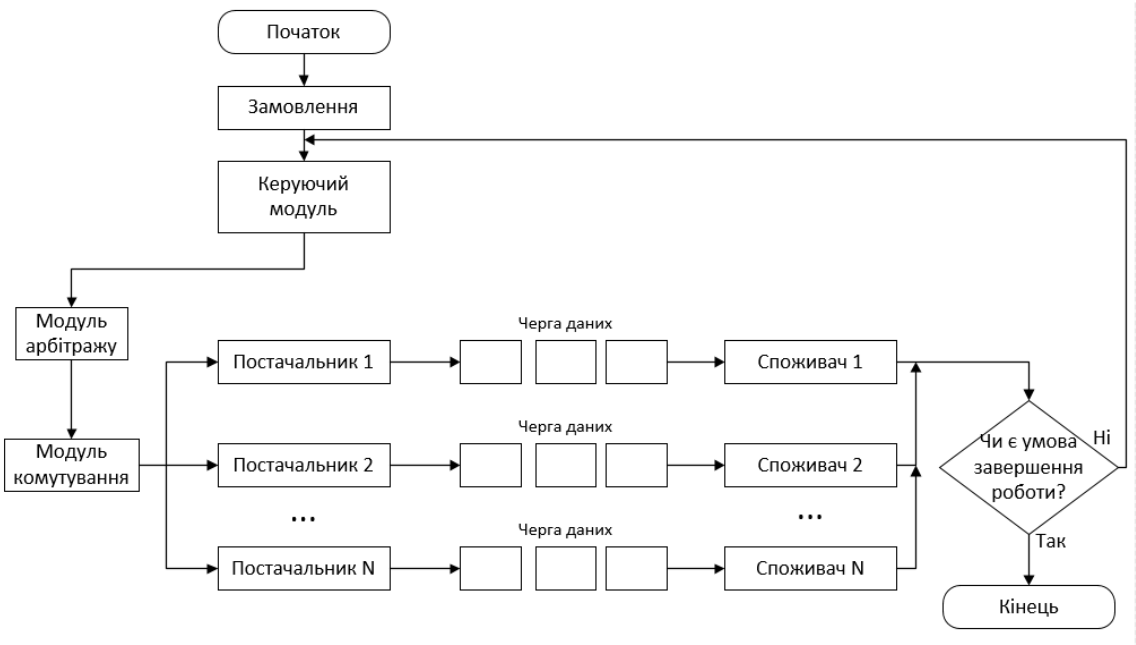


Рис. 3. Схема алгоритму роботи декількох постачальників і декількох споживачів із окремими чергами даних

Рис. 3 ілюструє роботу невеликої системи, яка може працювати на смартфоні, планшеті, комп'ютері тощо. Ця система виконуватиме замовлення, які їй надходять, вона визначає, якого типу це замовлення, скільки пам'яті потрібно виділити, скільки використати постачальників та споживачів, який запит сформувати до інтернет-ресурсу та інші параметри. В алгоритмі використано арбітраж, що дає змогу, своєю чергою, правильно розставити пріоритети при завантаженні файлів із зовнішнього ресурсу. Це можна застосовувати для отримання із зовнішнього ресурсу різних файлів, наприклад, аудіозаписів, відеофайлів, фотографій тощо і подавати їх на подальшу обробку. Програмна реалізація – за аналогією із підпрограми 1 та 2.

Програмну реалізацію для варіанта "один постачальник – один споживач" досліджено на прикладі мобільного клієнта для сайту imgur.com – сайт, який містить картинки, які можуть давати різні зареєстровані користувачі. Цей сайт надає вільний доступ до API, який використовується в програмі. Для відкриття цього додатка потрібно натиснути на іконку за назвою "ImgurMobile", як показано на рис. 4. Після цього відкриється головне вікно програми (рис. 5).

Головне вікно програми інформує про підвантаження даних з Інтернету стосовно сторінки, на якій зараз знаходиться користувач. Як тільки користувач прогорне вниз, то з'явиться нова картинка, яка почне одразу завантажуватись (рис. 5, *а*). Також можлива ситуація, коли потрібно одночасно завантажити декілька картинок, як це показано на рис. 5, *б*.

У програмі є інтерфейс з двома функціями:

– (void) addObject: (id) object;

– (id) getObject;

При завантаженні даних з Інтернету процедури, які обробляють відповіді, шо приходять з сервера, містять ці дві функції. Спочатку викликається функція addObject, яка отримує та записує дані з Інтернету в проміжний буфер, після чого викликається функція getObject, яка видобуває завантажені дані з буфера та подає їх на подальшу обробку.

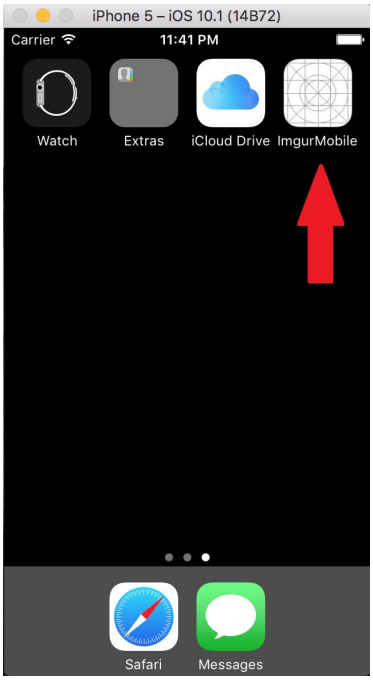


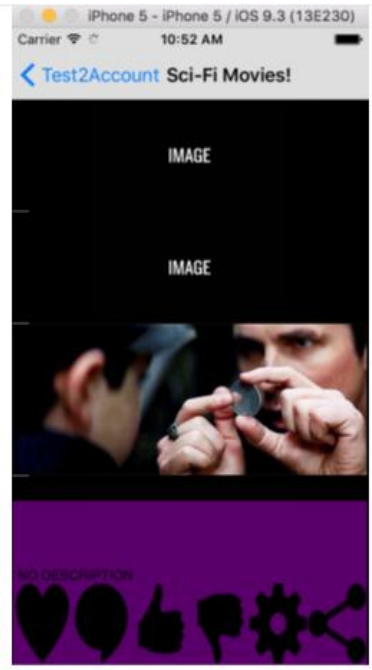
Рис. 4. Дисплей мобільного телефону

а

б

Рис. 5. Скріншоти головного вікна програми

З головного вікна програми користувач може переходити на інші сторінки та здійснювати пошук за певними критеріями картинок. Однією з функцій також є детальний перегляд матеріалів, закачаних із сервера, тобто користувач вибирає альбом, який хоче переглянути, натискає на відповідний покажчик, після чого відкривається нове вікно вже з детальним вмістом матеріалу (приклад на рис. 6). Повний вміст не відображається одразу – необхідний певний час для завантаження даних. Якщо завантажити весь матеріал одразу при запуску програми, це буде не ефективне використання пам’яті пристрою.



а

б

в

Рис. 6. Скріншоти вікна для детального перегляду фотографій, альбомів тощо

На рис. 6 зображено процес завантаження картинок у довільному порядку, тобто спостерігається поступове приймання фотоматеріалу. Спочатку завантажилась картинка, зображена на рис. 6, *а*, після неї наступна, тепер вже є дві картинки – рис. 6, *б*, і остання картинка – рис. 6, *в*. На деяких місцях стоїть звичайний заповнювач – "IMAGE", на місці якого буде картинка, як тільки вона завантажиться. Слід звернути увагу, що завантаження не обов'язково повинно виконуватись в послідовному порядку. В цьому прикладі можна спостерігати синхронізацію програми на мобільному пристроєві, де забезпечено коректне виконання всіх потоків у певному порядку. Коли користувач зробив певні запити на одержання картинок, він отримав відповідну відповідь на них. Використовуючи вищенаведені засоби, можна спостерігати повне завантаження контенту без втрати даних.

Висновки

Використовуючи запропоновані алгоритмічно-програмні засоби для синхронізації потоків, можна ефективно обмінюватися великими обсягами інформації між зовнішнім ресурсом (інтернетом, сервером) та комп'ютером чи комп'ютерною системою. Наведені алгоритми є універсальними для різних операційних систем.

1. Томас Бройнль. Паралельне програмування / Томас Бройнль. – К., 1997. – С. 91–133. 2. Корнеев В. В. Параллельные вычислительные системы / В. В. Корнеев. – М., 1999. – С. 117–133. 3. About Threads [Електронний ресурс] / Apple. - Режим доступу: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Multithreading/AboutThreads/AboutThreads.html#//apple_ref/doc/uid/10000057i-CH6-SW2/. 4. Creating Threads [Електронний ресурс] / Apple. – Режим доступу: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ Multithreading/ CreatingThreads/ CreatingThreads.html#//apple_ref/doc/uid/10000057i-CH15-SW2/. 5. Synchronization in java [Електронний ресурс] / Javapoint. – Режим доступу: http:// www.javatpoint.com/synchronization-in-java/. 6. Thread Safety [Електронний ресурс] / Apple. - Режим доступу: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Multithreading/ThreadSafety/ThreadSafety.html#//apple_ref/doc/uid/10000057i-CH8-SW1 7. Яркун В. І. Методика підвищення ефективності інформаційної системи при передачі великих об'ємів даних / В. І. Яркун // Вісник Тернопільського національного економічного університету "Інженерія програмного забезпечення". – 2016. – С. 173–175.