УДК: 004.451.45, 004.451.8

М. І. Ковалик, Р. М. Камінський
Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж

ОРГАНІЗАЦІЯ БАГАТОПОТОКОВОСТІ НА МОБІЛЬНІЙ ПЛАТФОРМІ ANDROID

© Ковалик М. І., Камінський Р. М., 2014

Опрацьовано й описано основні проблеми організації багатопотокової роботи у системі Android. Розглянуто основні варіанти кооперації фонових потоків та головного потоку програми, а також висвітлено переваги і недоліки кожної з моделей взаємодії.

Ключові слова: Android, опрацьовувач, координатор, багатопотоковість, асинхронне завдання.

In the article the main problems of the multi-threading organization in Android are processed and described. The basic options for cooperation of background threads and the main UI-thread of the program are considered and the advantages and disadvantages of each model interaction are highlighted.

Key words: Android, Concurrency, Multithreading, Handler, Coordinator, Asynchronous Tasks.

Вступ. Загальна постановка проблеми

Сучасні мобільні платформи володіють доволі потужними обчислювальними можливостями для розпаралелювання задач, тобто виконання незалежних одна від одної задач одночасно. Все частіше з'являються мобільні пристрої з 2-, 4- і 8-ядровими процесорами. Однак паралельна робота має певні обмеження, оскільки, якщо її організовано неправильно, користувач не відчує різниці в швидкості роботи між звичайними одноядровими та багатоядровими процесорами. Також у разі багатопотокової роботи збільшується ймовірність помилок одночасного опрацювання спільних даних, що може призвести до їх неправильності. Для мобільних пристроїв з системою Android характерним є особливий підхід до організації багатопотоковості та взаємодії потоків з інтерфейсом користувача. Система Android є порівняно новою платформою і характеризується недостатньою кількістю досліджень щодо загальноприйнятих термінів та технік стосовно правильної організації багатопотокової роботи у цій системі. Ця стаття актуальна, оскільки містить систематизацію та узагальнення вже існуючих досліджень у цій області, характеристику кожної з технік реалізації багатопотоковості, а також виділяє переваги і недоліки відповідного варіанта організації багатопотоковості.

Аналіз останніх досліджень та публікацій

Поняття паралельності нерозривно пов'язано з поняттям потоків. Потік – це найменша послідовність інструкцій, що може бути опрацьована незалежно одна від одної. Коли запускається програма Android, створюється один "*головний*" потік, який відповідає за диспетчеризацію і перенаправлення подій до відповідних компонентів інтерфейсу, взаємодію з користувачем, створення та відображення візуальних компонентів системи – так званий *потік інтерфейсу користувача* (*User Interface Thread* – UI- потік). Будь-які зміни до цих компонентів повинні бути здійснені у цьому потоці. Паралельно з головним можуть працювати і фонові потоки, що виконують будь-яку роботу, але не можуть змінювати стану компонентів інтерфейсу користувача. Це пов’язано з тим, що набір інструментів для роботи з користувачем інтерфейсу (Android UI Toolkit) не є потокобезпечним. Якщо ж зміни вносяться з будь-якого іншого потоку, виникає виняток – CalledFromWrongThreadException[1].

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Будь-які довготривалі дії, такі як доступ до мережі або до бази даних, необхідно здійснювати в окремому потоці. Це пов'язано з тим, що довготривалі дії у головному потоці можуть його блокувати, а оскільки UI-потік відповідає за взаємодію користувача і компонентів і буде заблокований на час виконання роботи, система не відповідатиме на будь-які дії користувача, такі як дотик до екрана, натиснення на кнопку або перевертання пристрою. У користувача може скластись враження, що програма «зависла». Якщо головний потік заблокований протягом 5 с, Android надсилає діалог "Програма не відповідає" ("Application not responding"). У цьому діалозі користувач може примусово завершити поточну програму [4].

І навіть більше, у останніх версіях Android, а точніше, починаючи з версії Honeycomb(3.0), доступ з головного потоку до мережі Інтернет заборонений. У разі такого доступу виникає виняток **NetworkOnMainThreadException**[6].

Формулювання мети

Здійснити аналіз доступних методів і особливостей багатопотокової роботи у системі Android.

Аналіз отриманих наукових результатів

Для уникнення можливих ситуацій блокування головного потоку в Android існує декілька варіантів багатопотокової організації роботи.

1. Використання потоків і механізму Опрацьовувача повідомлень (Handler), Повідомлення (Message), Виконуваного завдання (Runnable) (HaMeR).
2. Використання асинхронних завдань.

Кожен з цих механізмів характеризується певними перевагами та недоліками, особливостями роботи та специфікою організації.

Використання потоків і механізму Опрацьовувач повідомлень (Handler), Повідомлення (Message), (Виконувач) Runnable

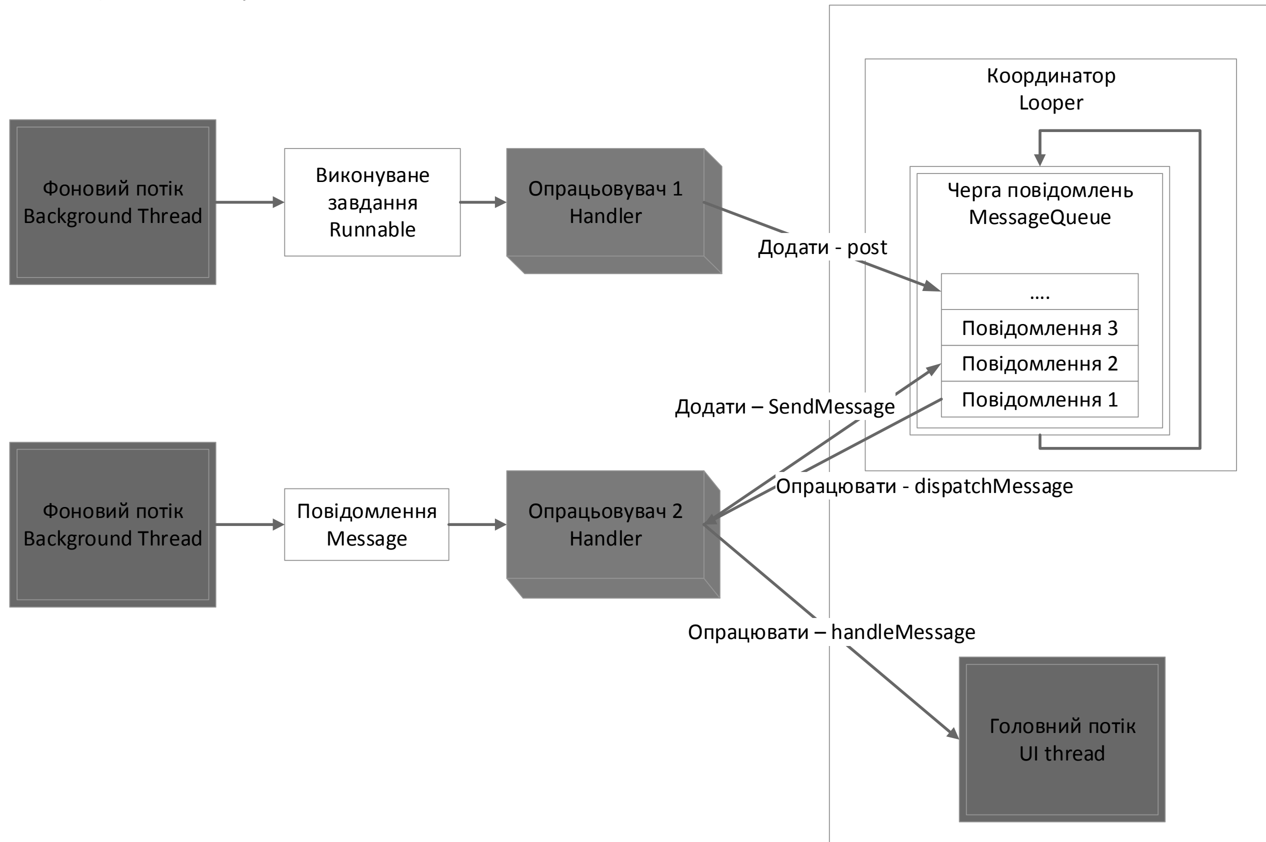

Рис. 1. Схематичне зображення кооперації компонентів моделі

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Першим можливим варіантом організації багатопотокової роботи є так звана модель **HaMeR** (HAndler, MEssage, Runnable). Ця модель складається з набору класів – Координатор (Looper), Опрацьовувач повідомлень (Handler), Черга Повідомлень (MessageQueue), Повідомлення (Message).

1. **Координатор (Looper)** – компонент, який здійснює переспрямування всіх повідомлень з Черги повідомлень до відповідних Опрацьовувачів повідомлень. Для потоку дозволений лише один Координатор. Якщо у потоці вже є створений Координатор і потік пробує ще раз здійснити його ініціалізацію, то виникає виняток часу виконання – **RuntimeException** з повідомленням, що лише один Координатор може бути створений (“Only one Looper may be created per thread”)[3].

Кожен потік може мати свій Координатор, тому існуватимуть декілька Координаторів одночасно, які можуть співпрацювати за допомогою Опрацьовувачів.

Для повноцінної роботи зв’язки класів Потік – Координатор – Опрацьовувач обов’язкове виконання таких кроків:

– створення і прив’язка Координатора до потоку за допомогою статичного методу `Looper.prepare();`

– створення Опрацьовувача і реалізація методу `void handleMessage(Message msg);`

– початок оброблення і перенаправлення повідомлень з Черги повідомлень, тобто запуск Координатора статичним методом – `Looper.loop();`

2. **Опрацьовувач (Handler)** – багатофункціональний компонент, який дає змогу опрацьовувати Повідомлення та виконувати Завдання в потоці, в якому був створений. Методи, які використовує Опрацьовувач, можна поділити на чотири категорії:

a. Публікація/видалення Виконуваних завдань.
b. Відправка повідомлень.
c. Отримання повідомлень.
d. Диспетчеризація і опрацювання повідомлень.

3. **Черга повідомлень (MessageQueue**) – синхронізована черга, у якій містяться Повідомлення, які потрібно перенаправити Координатором до відповідного Опрацьовувача.

4. **Повідомлення (Message)** – клас, призначений для передавання даних (*Bundle*) або виконуваних завдань (*Runnable*) від будь-якого потоку до потоку, в якому створений Опрацьовувач. Повідомлення складається з таких компонентів:

`Bundle data` – дані, що передаються у Повідомленні;

`Handler target` – Опрацьовувач, якому передається Повідомлення;

`Runnable callback` – Виконуване завдання, що передається Опрацьовувачу;

`int what` – число цілого типу для ідентифікації конкретного Повідомлення.

Зазвичай Повідомлення використовується або для передавання даних, або для запуску Виконуваного завдання в потоці Опрацьовувача.

Коли у фонового потоку з’являється необхідність передати дані у головний потік програми або у будь-який інший потік, то за допомогою методу `obtainMessage()` здійснюється підготовка Повідомлення. Можливі варіанти як для передавання порожнього повідомлення (для сповіщення про початок/завершення певних дій), так і для передавання певних результатів опрацювання до Опрацьовувача (результати роботи фонового потоку). В останньому випадку використовується метод `setData(Bundle)`, який прив’язує дані до конкретного екземпляра Повідомлення. У двох випадках Повідомлення передається за допомогою методу `sendMessage(Message)`. Після того, як Повідомлення передане, воно додається у Чергу Повідомлень Координатора. Координатор, дійшовши до цього повідомлення, перенаправляє Повідомлення до відповідного Опрацьовувача і Повідомлення опрацьовується в методі `handleMessage(Message)` у потоці, в якому створений Опрацьовувач. Послідовність кооперації компонентів зображено на рис. 2.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

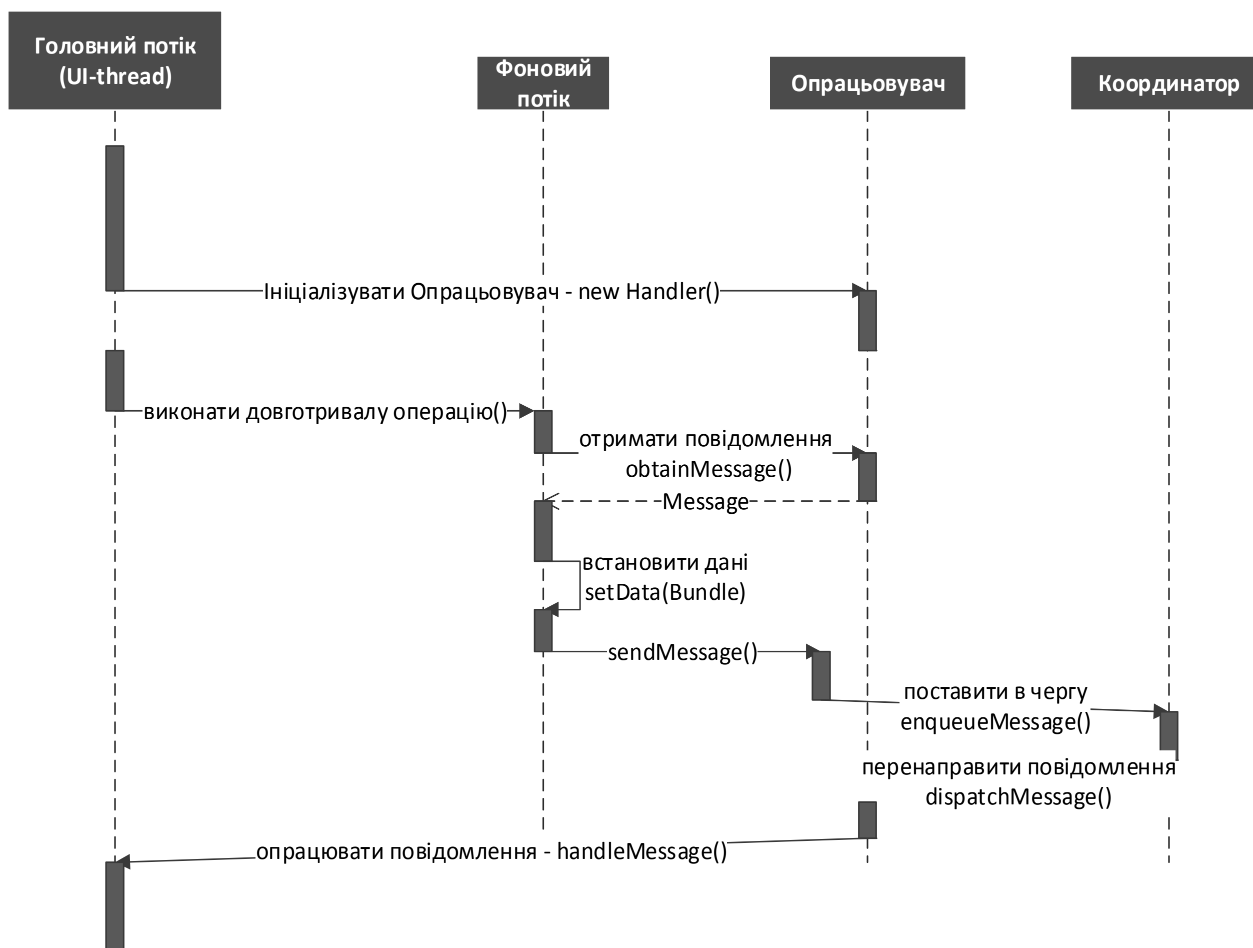

Рис. 2. Послідовність кооперації компонентів моделі

Аналогічно, фоновий потік може передати Виконуване завдання (Runnable) Опрацьовувачу. У цьому випадку за допомогою методу `post(Runnable)` Завдання, яке передане параметром, буде виконане в потоці, в якому створений Опрацьовувач. Для виконуваного Завдання немає потреби реалізовувати спеціальні методи Опрацьовувача або інші додаткові дії.

Організація HaMeR у Android

Після запуску програми система прив'язує Координатор до головного потоку програми, викликавши метод prepareMainLooper(). Цей метод створює єдиний головний Координатор, який працює доти, доки працює програма. Оскільки головний Координатор вже створений, повторно цей метод викликати не потрібно.

Черга повідомлень допомагає серіалізувати доступ до головного потоку і гарантує, що будь-які компоненти, створені в ньому, мають доступ до інших об'єктів, що теж створені в цьому потоці.

Одразу після створення Координатора створюється і Опрацьовувач для головного потоку відповідної Активності. Для полегшення роботи з головним потоком у Активності є метод `runOnUiThread(Runnable)`, що виконує Завдання, передане параметром Runnable. Цей метод є обгорткою, всередині якої відбувається простий виклик методу `post(Runnable)` для Опрацьовувача головного потоку.

Для випадків, коли необхідно створити Координатор для потоку, Android має готовий допоміжний клас HandlerThread, який має вже створений Координатор. Особливістю цього класу є те, що він має готовий метод `onLooperPrepared()`, який викликається, коли створений Координатор для цього потоку. У цьому методі доцільно створювати Опрацьовувач повідомлень [2].

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

AsyncTask

Модель асинхронних повідомлень є своєрідною спрощеною обгорткою моделі HaMeR і використовує її для міжпотокової роботи. Особливістю є використання готових шаблонних методів для полегшення роботи у потоках. До таких методів належать метод `doInBackground()`, `onPreExecute()`, `onPostExecute()`, `onProgressUpdate()`. Методи `onPreExecute()`, `onPostExecute()`, `onProgressUpdate()` виконуються в головному потоці і призначені для відображення змін візуальних компонентів інтерфейсу користувача, `doInBackground()` виконується у фоновому потоці, щоб запобігати блокуванню головного потоку.

`onPreExecute()` і `onPostExecute(Result… result)` викликаються до і після `doInBackground()` відповідно. `onPublishProgress(Progress… progress)` викликається лише після виклику методу `publishProgress(Progress… progress)`. Послідовність викликів зображено на Рис. 3

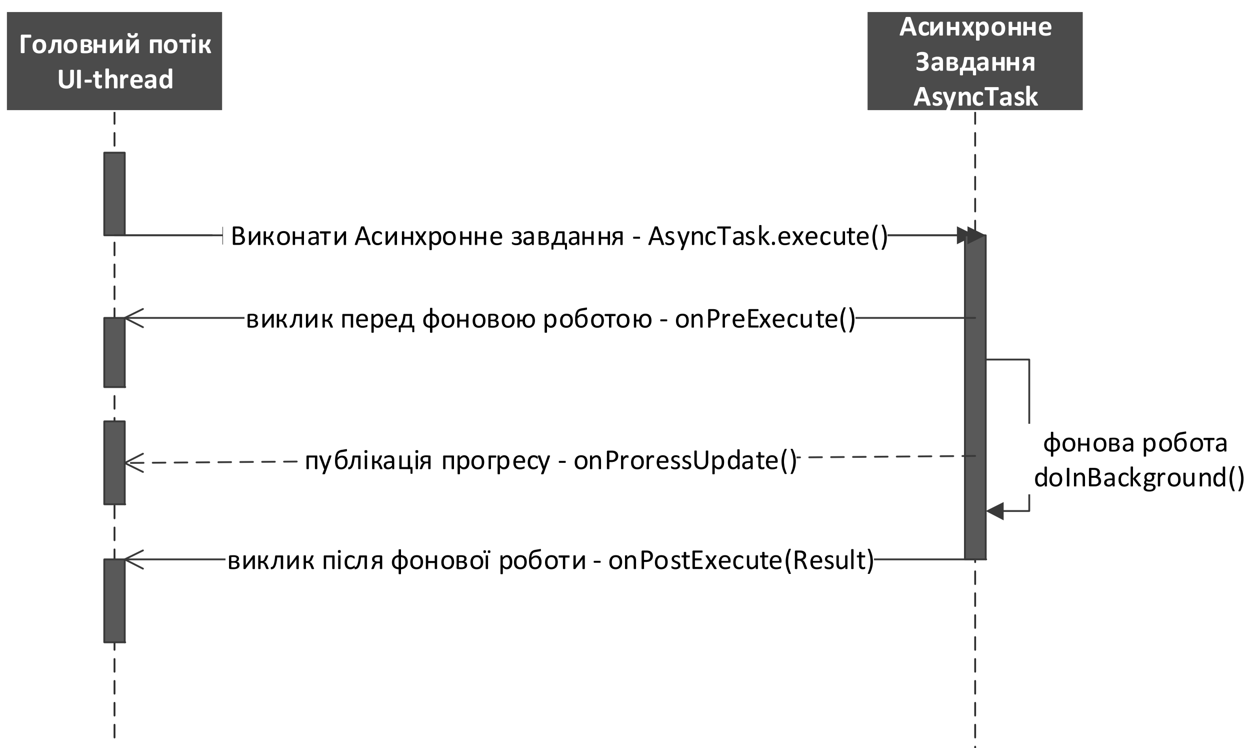

Рис. 3. Послідовність викликів методів

Асинхронні повідомлення використовують загальні типи (generics) для надання гнучкості під час передавання параметрів у Асинхронне завдання. Під час створення класу передається декілька уточнювальних типів: тип вхідних параметрів, тип параметрів для публікації та тип результату Завдання:

```
class AsyncTask extends AsyncTask<Params, Progress, Result>
Params – тип для передачі вхідних параметрів
Progress – тип для передачі параметрів публікації
Result – тип для передачі результатів опрацювання Завдання
```

Запуск Асинхронного завдання здійснюється з відповідним типом вхідних параметрів – `execute(Params… params)`. Params вказує тип вхідних даних і передається потім для опрацювання у фоновий метод `doInBackground (Params… params)`, який призначений для реалізації будь-якої довготривалої роботи. У разі довготривалої роботи доцільно відображати поточний прогрес у візуальний інтерфейс користувача для контролю і орієнтування тривалості

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

роботи. Для кожного відображення прогресу публікація прогресу відбувається за допомогою методу `publishProgress(Progress… progress)` з параметрами `Progress`. Асинхронне Завдання перенаправляє опрацювання прогресу у головний потік у метод `onProgressUpdate(Progress… progress)` для відображення змін у інтерфейсі користувача.

Важливим моментом є скасування завдання. Скасування потоків у Java є доволі складним процесом, часто може спричиняти помилки і неправильну роботу програми. Оскільки `doInBackground()` виконується у фоновому потоці, організація правильного скасування теж має певні особливості. Складністю є непряме завершення потоку – потік після отримання команди завершення не завжди зразу завершується і може виконуватись ще деякий час. Єдиним варіантом безпечного завершення виконання потоку є вихід з головного методу `run()`. Для вирішення цієї проблеми існує декілька підходів:

- Використання змінної для ідентифікації поточного стану потоку і періодична (у безмежному циклі) перевірка цієї змінної на стан завершення потоку. В такому разі жодні виклики до методів читання, очікування, приєднання чи блоків переривання не здійснюються, що може ускладнити процес завершення потоку.

- Використання методу `interrupt()`, який перериває виконання потоку. Для методів блокування, таких як `join()`, `sleep()`, `wait()` виникає виняток `InterruptedException`. Для перевірки завершення роботи у класі `Thread` існує також метод `interrupted()` та `isInterrupted()`, що повертають стан переривання роботи потоку. Відмінність полягає у тому, що перший встановлює значення змінної у попередній стан, а інший – ні.

Для Асинхронного завдання існує спеціальний метод `cancel(boolean mayInterruptIfRunning)`, який завершує виконання поточного Завдання, а для індикації стану завдання – `isCancelled()`. Якщо використовується цей метод, існує необхідність у періодичній перевірці (якщо робота відбувається у циклі) або перевірці у ключових місцях (якщо завдання виконується послідовно) методу `doInBackground()` стану завдання для швидкого виходу з методу у разі потреби. Якщо скасування Завдання відбулось під час виконання `doInBackground()`, то замість методу `onPostExecute(Result… result)` викличеться метод `onCancelled(Result… result)`.

Переваги і недоліки моделей

У кожної моделі є певні обмеження, які слід враховувати. Недоліки моделі HaMeR такі:

- Щоб отримати Повідомлення, потрібно завжди зберігати екземпляр класу Опрацьовувача.
- Складність реалізації.
- Для передавання складних даних через Повідомлення необхідна реалізація інтерфейсу Parcelable.

До переваг належать:

- Гнучкість.
- Необов'язковість роботи і викликів з головного потоку. Можливе використання для взаємодії декількох фонових потоків [5].

Недоліки моделі Асинхронних завдань:

- Створення екземплярів класу `AsyncTask`, запуск Асинхронного завдання (виклик методу `execute(Params…)` повинні здійснюватись з головного потоку.
- Асинхронне завдання може бути викликане лише один раз. Під час другого запуску виникає виняток.
- Різна поведінка у різних версіях Android. В початкових версіях (до API 1.6) всі асинхронні завдання виконувались послідовно в одному фоновому потоці. Ця поведінка змінилась у версії Android Donut (API 1.6). Тоді кожне асинхронне завдання виконувалось в окремому потоці. Це створювало проблеми із синхронізацією і одночасністю. І, починаючи з версії Android Honeycomb (API 3.0), асинхронні завдання можуть виконуватись як паралельно, так і послідовно.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Переваги Асинхронного Завдання:

- Простота використання
- Простота комунікації з головним потоком [7].

Висновки і перспективи подальших наукових розвідок

Система Android підтримує декілька моделей для реалізації співпраці з головним потоком. Модель HaMeR надає гнучкіший механізм обробки Повідомлень, надсилання Виконуваних завдань, дає змогу працювати не лише з головним потоком, а і з фоновими, але є складнішою у реалізації. Цю модель краще використовувати для часто повторюваних завдань. Модель Асинхронних завдань є спрощеним варіантом моделі HaMeR і призначена тільки для роботи з головним потоком системи. Цю модель доцільніше використовувати для довготривалої одноразової роботи або відображення прогресу певної роботи.

1. Processes and Threads [Electronic Resource]:[site] // Official documentation. – Mode of access: URL: http://developer.android.com/guide/components/processes-and-threads.html#Threads. – Title from the screen. – Last access:10.08.2014. 2. HandlerThread // Official documentation. – Mode of access: URL: http://developer.android.com/reference/android/os/HandlerThread.html. – Title from the screen. – Last access:10.08.2014. 3. Android Looper[Electronic Resource]:[videolecture] // Pattern-Oriented Software Architectures: Programming Mobile Services for Android Handheld Systems. – Mode of access: URL: https://class.coursera.org/posa-002/lecture/63. – Title from the screen. – Last access:10.08.2014. 4. Communicating with the UI Thread[Electronic Resource]:[site] // Official documentation. – Mode of access:URL: http://developer.android.com/training/multiple-threads/communicate-ui.html#Handler. – Title from the screen. – Last access:11.08.2014. 5. Handler vs AsyncTask[Electronic Resource]: [site]. – Mode of access:URL: http://stackoverflow.com/questions/2523459/handler-vs-asynctask. – Title from the screen. – Last access:11.08.2014. 6. NetworkOnMainThreadException[Electronic Resource]: [site]. – Mode of access:URL: http://developer.android.com/reference/android/os/NetworkOnMainThread Exception.html. – Title from the screen. – Last access:11.08.2014. 7. Sanjeev A. Deep dive into android AsyncTask[Electronic Resource]: [slides] // Bangalore Android User Group. – P. 14. – Mode of access:URL: http://www.slideshare.net/blrdroid/internals-of-asynctask. – Title from the screen. – Last access:11.08.2014. 8. Yehuda A. Android – Multithreading in a UI environment [Electronic Resource]: [site] // aviyehuda.com. – Mode of access:URL: http://www.aviyehuda.com/blog/ 2010/12/20/android-multithreading-in-a-ui-environment. – Title from the screen. – Last access: 11.08.2014. 9. Göransson A. Efficient Android Threading[Text].- Sebastopol:O'Reilly Media, 2014. – 260 p.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua