Обловацька М.В.
студент,
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ВИТІК ПАМ'ЯТІ В ANDROID

Для виконання своєї роботи кожному додатку потрібен такий ресурс як пам'ять. Щоб впевнитися, що для кожної програми є достатньо пам'яті, система Android повинна ефективно керувати розподілом існуючої пам'яті. У моменти, коли пам'яті не вистачає Android запускає збирання сміття (англ. garbage collector), який видаляє об'єкти, котрі вже не будуть використовуватися додатками. Принцип роботи збирача сміття наступний: є граф, в якому кожен існуючий об'єкт – це вершина, а посилання від будь-якого об'єкта на будь-який інший об'єкт – ребро. Деякі вершини на цьому графі – особливі. Це корені збирача сміття (англ. garbage collection roots) – ті сутності, які створені системою і продовжують своє існування незалежно від того, посилаються на них інші об'єкти чи ні. Якщо на графі існує будь-який шлях від даного об'єкта до будь-якого кореня, об'єкт не буде знищений збирачем сміття. Далі об'єкти, які залишилися, перевпорядковуються.

Тобто все, що обслуговує користувача, повинно зберігатись у пам'яті, а все інше вилучається з пам'яті, щоб звільнити ресурси.

Проте, коли код погано написано, об'єкти, що не використовуються можуть якимось чином посилатися на доступні об'єкти, тому збирач сміття буде позначати об'єкти, що не використовуються, як корисні і не зможе їх видалити. Це називається витік пам'яті.

Насправді жоден об'єкт не повинен лишатися в пам'яті довше, ніж потрібно, адже він займає цінний ресурс, який може бути необхідним для вирішення важливіших задач. Конкретно для Android витік пам'яті спричиняє наступні проблеми:

1. Буде доступно менше пам'яті. В результаті система Android частіше запускатиме збирач сміття. Це означає, що рендерінг користувацького інтерфейсу та обробки подій буде припинено. Тобто час завантаження додатку в Android збільшиться і користувачі будуть сприймати це як зависання додатку.

2. Якщо додаток має витік пам'яті, він не може вимагати пам'ять від невикористаних об'єктів. В результаті, він попросить у системи Android більше пам'яті. Проте існують певні обмеженні і система врешті-решт відмовиться виділяти пам'ять для додатку. Коли це станеться, користувач отримає збій пам'яті.

3. Вирішення проблеми витоку пам'яті в основному важко знайти в QA чи тестуванні, адже його важко відтворити. І звіт про аварійне завершення роботи, зазвичай, важко обґрунтувати, оскільки це може статися в будь-який час і де завгодно, коли віддача пам'яті отримує відмову від Android.

Існують способи пошуку витоків пам'яті, проте вони вимагають гарного розуміння того, як працює збирач сміття. Але в Android є багато корисних

інструментів, які допоможуть визначити можливі витоки або переконатися, що виявлено витік, коли якийсь код виглядає підозрілим.

1. Leak Canary від Square є хорошим інструментом для виявлення витоків пам'яті у додатку. Він створює слабкі посилання на процеси у додатку. Потім він перевіряє, чи буде посилання видалено після запуску збирача сміття. Якщо ні, він скидає купу (англ. heap) в файл та аналізує його, щоб підтвердити, чи є витік. Якщо так, то він показує сповіщення та в окремому додатку дерево довідки про те, як відбувається витік.

2. Android Studio має зручні інструменти для виявлення витоків пам'яті. Якщо існує підозра, що фрагмент коду у додатку може призвести до витоку активності, потрібно скомпілювати та запустити додаток на емуляторі чи пристрої та перейти до підозрілої активності, а потім повернутися до попередньої активності і перейти до розділу Memory в Android Studio. Перейшовши до Dump Java Heap можна автоматично виявити витоки, використовуючи Analyzer Tasks або переключити режим перегляду на вікно дерева пакетів і знайти активність, яку потрібно знищити.

Можна виділити наступні схеми витоків:

1. Витік в активності до статичного посилання. Статичне посилання зберігається до тих пір, поки ваш додаток в пам'яті. Активність має життєвий цикл, зазвичай знищується та повторно створюється протягом життєвого циклу додатка. Якщо існує пряме чи непряме статичне посилання в активності, то вона не може бути зібрана збирачем сміття. Активність може коливатися від кількох кілобайт до багатьох мегабайт залежно від того, який вона має вміст. Якщо активність має велику ієрархію перегляду або зображення з високою роздільною здатністю, це може спричинити витік великої кількості пам'яті.

Можна розрізнити наступні витоки пам'яті в цій категорії:

– витік в активності до статичного елементу інтерфейсу;
– витік в активності до статичної змінної;
– витік в активності до одного об'єкта;
– витік в активності до статичного екземпляру внутрішнього класу діяльності.

2. Витік в активності в робочий потік. Робочий потік також може вимикати активність. Якщо існує пряме чи непряме посилання на активність з робочого потоку, який працює довше за активність, можна також отримати витік пам'яті.

Маємо наступні витоки в цій категорії:

– витік в активності до потоку;
– витік в активності до обробника;
– витік в активності до AsyncTask.

3. Витік потоку в себе. Кожного разу, коли розробник відкриває робочий потік з активності, він несе відповідальність за управління цим потоком. Оскільки робочий потік може існувати довше, ніж активність, потрібно зупинити робочий потік, коли активність вже знищена.

Отже, було розглянуто поняття витоку пам'яті та процеси, які до нього призводять та наслідки, які виникають в системі Android. Також представлено

два інструменти для боротьби з витоками пам’яті та розглянуто їх загальні схеми в Android.

Список використаних джерел:

1. Keeping Your App Responsive [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.android.com/training/articles/perf-anr.html

2. Java Memory [Електронний ресурс]. – Режим доступу до ресурсу: https://www.dynatrace.com/resources/ebooks/javabook/how-garbage-collection-works/

3. HPROF Viewer and Analyzer [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.android.com/studio/profile/am-hprof.html

4. Finally understanding how references work in Android and Java [Електронний ресурс]. – Режим доступу до ресурсу: https://medium.com/google-developer-experts/finally-understanding-how-references-work-in-android-and-java-26a0d9c92f83#.h9w7hp13h

Парасочка Є.В.

студент,
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ВИКОРИСТАННЯ ПРИСТРОЇВ STATCOM З МЕТОЮ ПІДВИЩЕННЯ ДИНАМІЧНОЇ СТІЙКОСТІ ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМ

Сучасні електроенергетичні системи(ЕЕС) є складними динамічними системами з глибокими взаємними зв’язками. Для розв’язку задач керування і захисту в сучасних ЕЕС використовуються мікроелектронні пристрої, комп’ютери та високо швидкісні канали передачі даних. Але швидкодія силових керуючих в сучасних ЕЕС визначається інерційністю механічних перемикачів. Інтенсивний розвиток силової електроніки на базі IGBT(Insulated Gate Bipolar Transistor) дало можливість створювати транзистори здатні працювати в силових мережах, а на їх основі – технології керованих гнучких електропередач змінного струму або FATCS(Flexible alternating current transmission systems).

Основне завдання технології FACTS полягає в підвищенні ефективності управління потоками потужності, регулювання напруги, забезпечення статичної або динамічної стійкості. Така можливість забезпечується завдяки здатності елементів FACTS управляти взаємопов'язаними параметрами, що визначають функціонування магістральних ЛЕП, такими, як повний опір, струм, напруга, кут фазового зсуву між напругою на кінцях ЛЕП, згасання коливань на різних частотах і т.д.

Основні елементи пристроїв FACTS є сімейство автоматичних пристроїв (регуляторів) великої потужності, кожний з яких може застосовуватися як індивідуально, так і у взаємодії з іншими пристроями для керування одним або більшим числом взаємозалежних параметрів електроенергетичних систем. Загальна структура елементів FACTS зображена на рис. 1 [4, с. 82].