АНАСТАСІЯ ВІКАРЧУК, ДЕНИС КАТЄЛЬНІКОВ

ВПРОВАДЖЕННЯ В НАВЧАЛЬНИЙ ПРОЦЕС ВИВЧЕННЯ ШАБЛОНІВ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ

Анотація

Розглянуто основні шаблони паралельного програмування, які необхідно впровадити в навчальний процес.

Abstract

The main concurrency patterns waiting to be implemented in the educational process, are considered in this paper.

Вступ

Шаблон проектування або патерн (англ. design pattern) у розробці програмного забезпечення - повторювана архітектурна конструкція, що представляє собою вирішення проблеми проектування в рамках деякого контексту, що часто виникає. При викладанні дисциплін “Шаблони проектування програмного забезпечення” та “Моделювання та аналіз програмного забезпечення” приділяється суттєва увага вивченню подібних конструктивних елементів, адже їх використання дозволяє формалізувати процес проектування програмного забезпечення, робить можливим застосування вдалих проектних рішень і розширює галузь застосування CASE засобів, що відчутно скорочує витрати. Зазвичай вивчення шаблонів проектування обмежується вивченням шаблонів “Gang of Four” (так звані GoF-шаблони) [1,2], шаблонів GRASP [3] та шаблонів архітектури програмного забезпечення [4]. Однак останнім часом завдяки зворотному зв’язку від IT-компаній стало зрозуміло, що в наведеному наборі шаблонів не вистачає ще одного виміру, а саме: шаблонів проектування паралельного програмування, знання яких необхідне випускникам IT-спеціальностей при розро-

бці високонавантажених backend – систем з розпаралелюванням роботи програмних модулів.

В даній роботі розглядаються основні шаблони паралельного програмування, які можуть бути впроваджені в навчальний процес.

Загальна інформація

Роботи над шаблонами паралельного програмування ведуться вже давно: можна згадати класичну працю[5] з відомої серії Pattern-Oriented Software Architecture (POSA), де розглядають саме питання конкурентного доступу до ресурсів при паралельному та мережевому програмуванні. В роботі [6] показано, як розробляти та впроваджувати підтримувані та ефективні паралельні алгоритми, використовуючи комбінований, структурований, масштабований та машинно-незалежний підхід до паралельних обчислень. В роботі [7], яка є найновішою, пропонуються нові підходи до побудови паралельних застосунків, в яких паралелізм повністю інкапсульований у самому шаблоні і не містить прихованих залежностей від інших частин системи.

Аналіз наведених робіт дозволяє виділити чотири найбільш уживаних шаблона проектування: Активний об’єкт (Active Object), Планувальник (Scheduler), Монітор (Monitor), Блокування з подвійною перевіркою (Double-checked locking).

Активний об’єкт (Active Object)

Шаблон проектування Активний об’єкт відокремлює виконання методу від виклику методу для об’єктів, кожен з яких знаходиться у своєму власному потоці керування[5]. Мета полягає в тому, щоб запровадити паралельність за допомогою асинхронного виклику методів і Планувальника (Scheduler) для обробки запитів[8].

Шаблон проектування складається з шести елементів[9]:

1. Проксі, який забезпечує інтерфейс для клієнтів із загальнодоступними методами.
2. Інтерфейс, який визначає запит методу для активного об’єкта.
3. Список нерозглянутих запитів від клієнтів.

4. Планувальник, який вирішує, який запит виконати наступним.
5. Реалізація методу активного об'єкта.
6. Зворотний виклик або змінна для отримання клієнтом результату.

Покращує паралелізм додатків і спрощує складність синхронізації: паралелізм покращено завдяки дозволу клієнтським потокам і викликам асинхронних методів запускатись одночасно. Складність синхронізації спрощується за допомогою Планувальника, який оцінює обмеження синхронізації, щоб гарантувати серіалізований доступ до Слуг (Servant), залежно від їх стану.

Прозоро використовуйте доступний паралелізм: якщо апаратні та програмні платформи ефективно підтримують кілька центральних процесорів, цей шаблон може дозволяти одночасному виконанню кількох активних об’єктів, залежно від їх обмежень синхронізації.

Порядок виконання методів може відрізнятися від порядку їх виклику: асинхронно викликані методи виконуються на основі їхніх обмежень синхронізації, які можуть відрізнятися від порядку їх виклику.

Планувальник (Scheduler)

Планувальник може мати на меті одну або декілька цілей, наприклад:

- максимізація пропускної здатності (загального обсягу виконаної роботи за одиницю часу);
- мінімізація часу очікування (час від готовності роботи до початку виконання першої команди);
- мінімізація затримки або часу відповіді (час від готовності роботи до її завершення у випадку пакетної діяльності [10] або доки система не відповість і не передасть користувачеві перше виведення інформації у разі інтерактивної діяльності)[11];
- максимізація справедливості (рівний час процесора для кожного процесу або більш загально відповідний час відповідно до пріоритету та робочого навантаження кожного процесу).

На практиці ці цілі часто конфліктують (наприклад, пропускна здатність проти затримки), тому планувальник реалізує відповідний компроміс. Перевага

вимірюється будь-яким із зазначених вище критеріїв, залежно від потреб і цілей користувача.

У середовищах реального часу, таких як вбудовані системи автоматичного керування в промисловості (наприклад, робототехніка), планувальник також повинен гарантувати, що процеси можуть відповідати вчасно; це має вирішальне значення для підтримки стабільності системи. Заплановані завдання також можна розповсюджувати на віддалені пристрої в мережі та керувати ними через адміністративну серверну частину.

Монітор (Monitor)

Монітор — це потокобезпечний клас, об'єкт або модуль, який обгорнутий навколо м'ютексу, щоб безпечно дозволити доступ до методу чи змінної більш ніж одному потоку. Визначальною характеристикою монітора є те, що його методи виконуються із взаємним виключенням: у кожен момент часу щонайбільше один потік може виконувати будь-який із його методів. Використовуючи одну або більше умовних змінних, це також може надати потокам можливість очікувати певної умови (таким чином, виконуючи роль «монітора»)[12].

Блокування з подвійною перевіркою (Double-checked locking)

Блокування з подвійною перевіркою (також відоме як «оптимізація блокування з подвійною перевіркою» [5]) — це шаблон розробки програмного забезпечення, який використовується для зменшення накладних витрат на отримання блокування шляхом тестування критерію блокування («підказка блокування») перед отриманням блокування. Блокування відбувається, лише якщо перевірка критерію блокування вказує на необхідність блокування.

Шаблон, реалізований у деяких комбінаціях мови/апаратного забезпечення, може бути небезпечним. Іноді це можна вважати антипаттерном.

Зазвичай він використовується для зменшення накладних витрат на блокування під час реалізації «ледачої ініціалізації» в багатопоточному середовищі, особливо як частина шаблону Одинак (Singleton). Лінива ініціалізація дозволяє уникнути ініціалізації значення до першого доступу до нього.

Висновки

Огляд літератури, присвяченої розробці шаблонів паралельного програмування дозволив визначити чотири найбільш уживані шаблони для додання їх в робочу навчальну програму дисциплін “Шаблони проектування програмного забезпечення” та “Моделювання та аналіз програмного забезпечення”.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns. Elements of Reusable Object-Oriented Software. Boston, Massachusetts:Addison-Wesley Pub Co., 1994. 416 p.

2. Васянович Є.А. , Катєльніков Д.І. Використання шаблонів проектування у сучасному підході до розробки програмного забезпечення. Матеріали конференції «L Науково-технічна конференція підрозділів Вінницького національного технічного університету (2021)», Вінниця, 2021. [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/author/submission/12552. Дата звернення: Листопад, 2022.

3. Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Pearson:London, 2004. 736 p.

4. Mary Shaw, David Garlan. Software Architecture. Perspectives On An Emerging Discipline. Hoboken, NJ:Prentice Hall, 1996. 264 p.

5. Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann. Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. Hoboken, NJ : John Wiley & Sons, 2000. 633 p.

6. Michael McCool, James Reinders, Arch Robison. Structured Parallel Programming: Patterns for Efficient Computation, 1st Edition. Burlington, Massachusetts:Morgan Kaufmann, 2012. 432 p.

7. Lucian Radu Teodorescu. Concurrency Design Patterns. Overload #159, October 2020. [Електронний ресурс]. Режим доступу:

https://accu.org/journals/overload/overload159. Дата звернення: Листопад, 2022. pp. 12-18.

8. Bass, L., Clements, P., Kazman, R. Software Architecture in Practice. Boston, Massachusetts : Addison Wesley, 2003. 512 p.

9. R. Lavender, D. Schmidt. Active Object: An Object Behavioral Pattern for Concurrent Programming. Pattern languages of program design 2. Boston, MA : Addison-Wesley Longman Publishing Co., Inc., 1996. pp. 483–499.

10. Leonard Kleinrock. Queueing Systems, Vol. 2: Computer Applications. Hoboken, NJ : Wiley-Interscience, 1976. 576 p.

11. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. Operating System Concepts, 9th edition. Hoboken, NJ : Wiley Publishing, 2012. 919 p.

12. Hoare, C. A. R. Monitors: an operating system structuring concept. Comm. ACM. 17 (10), October 1974: pp. 549–557.

Вікарчук Анастасія Вікторівна — студентка групи 3ПІ-19б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: vikarchukn2016@gmail.com

Катєльніков Денис Іванович - кандидат технічних наук, доцент кафедри програмного забезпечення, Вінницький національний технічний університет, м. Вінниця, e-mail: fuzzy2dik@gmail.com.

Vikarchuk Anastasiia Viktoravna - student gr. 3PI-19b, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: vikarchukn2016@gmail.com

Katielnikov Denys Ivanovych - PhD, Associate Professor of Software Engineering Department, Vinnytsia National Technical University, Vinnytsia, E-mail: fuzzy2dik@gmail.com