УДК 681.3

О.В. Корочкін, О.В. Русанова, О.М. Крутько

АНАЛІЗ ЗАСОБІВ УПРАВЛІННЯ ПОТОКАМИ В МАСШТАБОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМАХ

Анотація: В даній статті представлені результати аналізу засобів керування потоками в сучасних мовах і бібліотеках паралельного програмування. Розглянути мови Java, C#, Ada, Python, бібліотеки WinAPI, OpenMP.

Ключові слова: потоки, взаємодія потоків, масштабовані комп’ютерні системи.

Вступ

Сучасні комп'ютерні системи ґрунтуються на використанні багатоядерних процесорів, кількість ядер у яких безперервно зростає. На сьогодні стандартом є восьмиядерний процесор, масовими стають 12-ядерні процесори, на підході 32 і 64-ядерні процесори, вже представлені на ринку.

Використання систем з різною кількістю ядер ставить завдання розробки програмного забезпечення, яке ефективно виконуватиметься в системах з різною кількістю ядер (масштабованих системах).

Сучасні мови та бібліотеки паралельного програмування забезпечують різні засоби роботи з потоками [1-5], існуючі засоби удосконалюються, а також з'являються нові. Вибір засобів програмування потоків та засобів організації їх взаємодії багато в чому визначає ефективність паралельної програми для масштабованої комп'ютерної системи.

Представлений у роботі аналіз дозволяє вибрати оптимальні засоби роботи з потоками при розробці програмного забезпечення для паралельних систем. Завдання ускладнюється при розробці масштабованих програм, які повинні дозволити адаптувати їх за зміни структури комп'ютерної системи.

Потоки

Працюючи з потоками необхідно вирішити завдання створення необхідної кількості потоків, а також - завдання організації взаємодії потоків.

Перше завдання пов'язане безпосередньо з описом потоку (групи потоків), формуванням ідентифікатора потоку та пріоритету, вибору процесора для виконання, розміру стека потоку, дій потоку, запуску та завершення потоку.

Друге завдання є складнішим, оскільки виконання паралельної програми є критично залежним від організації взаємодії потоків. Невдала організація взаємодії потоків може стати джерелом неправильної роботи програми, збільшення часу

 ISSN 1560-8956

виконання програми, а також появи тупикових ситуацій (deadlocks), що може призвести до припинення виконання програми.

Для масштабованих програм вирішення обох завдань ускладнюється, так як кількість потоків може змінюватися. Більшою мірою це ускладнення стосується організації взаємодії потоків.

Розглянемо обидва завдання та проаналізуємо сучасні засоби для його вирішення.

Створення потоків

Можна відокремити кілька підходів до опису та створення потоків [5]:

- за допомогою використання спеціальних класів (модулів), які безпосередньо дозволяють описати дії та властивості потоку (Java, Ada);

- за допомогою потокових функцій, коли дії потоку задаються за допомогою функції, яка визначатиме поведінку потоку (C#, WinAPI);

- за допомогою виділення у послідовній програмі ділянок, які виконуватимуться паралельно (OpenMP);

- за допомогою створення копії всієї програми та паралельного виконання цих копій (MPI, PVM).

Додаткові можливості забезпечує об'єднання потоків у групи (пули) та використання черг різного виду, які дозволяють оптимізувати процес виконання потоків (Java, Python, Ada, C#, MPI).

Кожен підхід має свої переваги, мета яких - спростити розробку паралельної програми, її налагодження, модифікацію. Масштабування програми має дозволити змінювати (статично чи динамічно) кількість потоків, і навіть визначати особливості виконання.

Нижче наведено підхід до створення (опису) потоку масштабованої програми в мові Ada. Його особливістю є опис потоку як спеціального типу, а також використання дискримінанта (конструктора). Тип дозволяє створювати нові потоки (масиви потоків), а дискримінант – визначати особливості виконання кожного потоку залежно від значення параметра Idn:

```
task type T_Type(Idn: integer) is
      pragma Priority(7);                -- пріоритет потоку
      pragma Stack_Size (1000);          -- розмір стека потоку
      entry Send_T2(X: in integer);      -- засіб взаємодії  потоку
 end T_Type;
 T1: T_Type2(10);                        -- одиночний потік
 T100 is array (1..100)  of  T_Type(s);  -- масив  потоків.
```

Організація взаємодії потоків

Взаємодія потоків включає комунікацію (передачу даних між потоками) та синхронізацію потоків.

Реалізація цих дій залежить від моделі, обраної в мові для взаємодії: моделі, що базується на загальних змінних (shared variables model), або моделі, що базується на повідомленнях (message passing model). Більшість мов (бібліотек) орієнтовані на одну із зазначених моделей (як правило, першу), але є й такі, що підтримують обидві моделі [1, 5].

Вибір моделі, заснованої на загальних змінних, пов'язаний із необхідністю вирішення двох проблем: завдання взаємного виключення (mutual exclusion) та завдання синхронізації (synchronization).

Мова (бібліотека) має надати розробнику різноманітні засоби для ефективного вирішення зазначених проблем. Їх можна розділити на низько рівневі та високорівневі. Цікаво, що ряд мов (бібліотек) спочатку відмовлялися від підтримки низькорівеневих засобів, але пізніше поверталися до них (наприклад, поява семафорів у пізніших стандартах мов Java та Ada). З іншого боку, реалізація цих засобів, зазвичай, має особливості.

До низькорівневих засобів належать семафори, мютекси, події. Вони ефективні при незначній кількості потоків та ресурсів, але їх використання проблематично зі збільшенням кількості потоків у масштабованій програмі.

Завдання взаємного виключення

Вирішення завдання взаємного виключення передбачає контроль потоків при зверненні до загального ресурсу або контроль безпосередньо загального ресурсу [3, 5]. Перший вид контролю забезпечують семафори, мютекси, критичні секції, другий - монітори.

Семафори та мютекси представлені практично у всіх мовах та бібліотеках. Відмінності в реалізації незначні, пов'язані, як правило, з вибором типу семафору (бінарні або множинні) та його значень (логічні чи цілі), Масштабованість низькорівневих засобів ґрунтується на створенні додаткових семафорів, а також зміні безліч їх значень. Тут можливе використання масивів семафорів, що не завжди підтримується мовами або бібліотеками.

Важливо пам'ятати, що механізми семафорів і мютексів є джерелом тупиків, оскільки ґрунтується на операціях блокування потоків. Крім того, небезпеку створюють дії, пов'язані з переповненням допустимих значень семафорів. Все це вимагає ретельного контролю при використанні семафорів з боку розробника**.**

Критичні секції позбавлені недоліків семафорів та мютексів, що забезпечує більш надійний механізм вирішення завдання взаємного виключення. Реалізація механізму широко представлена у вигляді критичних секцій (WinAPI), синхронізованих

ISSN 1560-8956

блоків (Java), класу monitor C#. lock – конструкцій (C#, Python) [2, 4]. Однак їх використання для систем, що масштабуються, може викликати певні труднощі з точки зору їх тиражування для нових потоків і ресурсів.

Механізм моніторів забезпечує високорівневий підхід до вирішення завдання взаємного виключення в системах, що масштабуються. Монітор - програмний модуль, що інкапсулює загальний ресурс, що забезпечує його надійний захист. Особливий доступ до загального ресурсу забезпечують процедури монітора, які мають взаємовиключні властивості.

Найбільш ефективна реалізація концепції монітора виконана у мові Ада, де монітор представлений спеціальною мовною конструкцією – захищеним модулем (типом) protected [5]. Важливо, що захищений модуль дозволяє вирішувати як завдання взаємного виключення, так і завдання синхронізації. Для цього використовуються три види захищених операцій, які оптимізують дії з загальним ресурсом. Наявність у захищених входах спеціальних конструкцій - бар'єрів дозволяє реалізувати умовний доступ до загального ресурсу. Особливості масштабування забезпечуються через розміщення загального ресурсу в окремому модулі, описаному через захищений тип. Можливість керування чергами, що формуються при доступі до загальних ресурсів, також створюють передумови для оптимізації їх використання.

В мові Java відсутній певний клас-монітор. При побудові монітора інкапсуляцію загального ресурсу забезпечує модифікатор private, а взаємовиключний доступом до загального ресурсу - методи монітора з модифікатором synchronized.

Монітори дозволяють ефективно розробляти масштабовані програми, які можна налаштовувати на потрібну кількість процесорів у системі. Використання концепції моніторів набуло розвитку у мовах через появу вже готових моніторів. Прикладом такого монітора є клас AtomicInteger, який містить цілочисельну змінну і надає через методи класу понад двадцять атомарних операцій над нею [3].

Ще одним механізмом, що забезпечує роботу із загальними ресурсами, є використання atomic/volatile прагм, змінних та класів (C#, Ada, Java, Python).

Завдання синхронізації

Завдання синхронізації передбачає синхронізацію двох потоків (один потік чекає на подію в іншому), а також колективну синхронізацію (один потік чекає на події в декількох потоках, кілька потоків чекають на подію в одному потоку, група потоків чекає на події в іншій групі потоків).

Крім семафорів, синхронізацію потоків за допомогою низькорівневих засобів забезпечує використання механізму подій (events), реалізація яких представлена в WinAPI, C#. Їх використання ефективно реалізує множинну синхронізацію за динамічної зміни кількості взаємодіючих потоків.

Захищений модуль поряд з ефективним розв'язанням задачі взаємного виключення дозволяє реалізувати різні види задачі синхронізації. Бар'єри в захищених входах, можливість керування чергами, пов'язаними з входами, масиви входів дозволяють реалізувати гнучку систему очікування та обслуговування подій.

Слід зазначити інтерес до механізму бар'єрів, який забезпечує підтримку множинної синхронізації. Його реалізація є в OpenMP, а також з'явилася в пізніх версіях мов Ада та Java.

Таблиця 1.

Засоби	**Java**	**C#**	**Ada**	**Python**	**WinAPI**	**OpenMP**	**MPI**
Семафори	+	+	+	+	+		
Мютекси		+		+	+		
Події		+		+	+		
Критичні секції	+	+		+	+	+	
Бар’єри	+	+	+	+	+	+	+
Atomic/Volatile	+	+	+	+	+	+	
Монітори	+		+				
Черги	+	+	+	+			
Пули	+	+		+	+	+	
Повідомлення			+				+

Традиційні засоби організації взаємодії потоків базуються на припиненні потоків (засоби, що блокують), що збільшує час виконання програми.

Останнім часом було приділено увагу створенню засобів, що не блокують, які реалізують призупинення потоків без блокування. Це дозволяє відмовитися від виклику системних команд операційної системи з блокування та розблокування потоків. Такі механізми отримали реалізацію у мовах Java, C#, Python. [4].

У табл. 1 представлено наявність основних засобів взаємодії потоків у сучасних мовах та бібліотеках паралельного програмування.

Висновки

1. Сучасні бібліотеки та мови паралельного програмування забезпечують широкий спектр інструментів для роботи з потоками у паралельних системах.

2. Масштабовані паралельні програми характеризуються збільшенням та динамічною зміною кількості потоків у системі, що значно ускладнює організацію їх взаємодії.

3 Класичні низькорівневі механізми типу семафорів, мютексів, подій, а також критичних секції не повною мірою дозволяють організувати вирішення завдань синхронізації та взаємного виключення в системах, що масштабуються.

4. Для організації взаємодії потоків у системах, що масштабуються, найбільший інтерес представляють засоби, що реалізують концепцію моніторів, зокрема, механізм

ISSN 1560-8956

захищених модулів мови Ада, який також дозволяє об'єднати вирішення завдань взаємного виключення та синхронізації.

СПІСОК ВИКОРИСТНИХ ДЖЕРЕЛ

1. Barnes J. Programming in Ada 2012 with a Preview of Ada 2022, 2nd Edition. - Cambridge University Press; 2022. - 967 p.

2. Nagel Chrisian, Professional C# and .NET. Wrox; 2021. - 1008 p.

3. Oaks S. Java Performance: In-depth Advice for Tuning and Programming Java 8, 11, and Beyond. O’Reilly Media, Inc.; 2end Edition, 2020. – 452 p.

4. Gorelick M., Ozsvald I. High Performance Python, O’Reilly Media, Inc.; 2end Edition, 2020), - 470 p.

5. Жуков І., Корочкін О. Паралельні та розподілені обчислення. Навч.посібн. 2-е видання, Київ: «Корнійчук», 2014. - 284 с.