комп'ютерних засобів // Сучасні проблеми в комп'ютерних науках: Зб. наук. пр. – Львів: Держ ун-ту "Львівська політехніка", наукове товариство ім. Шевченка, 2000. – С. 58–67. 9. Черкаський М.В. SH-модель алгоритму // Вісн. Держ. ун-ту "Львівська політехніка". – Львів, 2001. 10. Черкаський М., Абдалла Саїд Садек. Псевдо SH-модель // Вісн. Нац. ун-ту "Львівська політехніка". – 2004. – № 523. – С. 145–150.

УДК 681.324

Б. Демида, А. Ратз*
Національний університет "Львівська політехніка",
кафедра автоматизованих систем управління,
*Львівська залізниця

ОПТИМІЗАЦІЯ АЛГОРИТМУ ПОШУКУ КЛЮЧІВ ІЄРАРХІЧНОГО ДЕРЕВА СИСТЕМНОГО РЕЄСТРУ WIN32 ІЗ ЗАСТОСУВАННЯМ МЕТОДІВ ПОТОКОВОЇ БАГАТОЗАДАЧНОСТІ

Зроблено спробу розроблення модифікованого алгоритму пошуку ключів ієрархічного дерева системного реєстру Win32 із використанням методу часового розпаралелювання процесу виконання операцій між множиною автономних потоків. Опис розробленого алгоритму містить також і порівняльну характеристику алгоритмів послідовного та паралельного виконання операцій пошуку з точки зору ефективності їхнього застосування. Ефективність описаного алгоритму визначають на основі розрахунку показника критичної тривалості виконання операцій пошуку ключів та процентної величини сумарного виграшу в часі завдяки використанню розробленого алгоритму.

The topic is being discussed in this technical article concerns the development of modified Win32 System Registry hierarchical tree key-node retrieval algorithm. According to the multithreading, the key-node retrieval operations are synchronously performed by the group of independent threads. The article's detailed review also includes the algorithm effectiveness evaluation which is based on determining the differences between either asynchronous or synchronous data retrieval process. The conclusion to the given material provides estimation of the overall productivity increasing percent ratio.

Вступ

Одними з найважливіших критеріїв оцінки ефективності впровадження сучасних програмних засобів оброблення великих масивів ієрархічних даних є час та швидкість виконання операцій [2]. До згаданих вище засобів насамперед належать і програмні засоби, орієнтовані на оброблення інформації, що зберігається в ієрархічних сховищах даних, зокрема в базі даних системного реєстру [2]. Головним недоліком існуючих сьогодні алгоритмічних розробок цих програмних засобів є невисокі показники продуктивності виконання операцій пошуку за достатньо великих обсягів інформації [2].

Серед існуючих методів підвищення продуктивності процесу оброблення даних найефективнішим було визнано метод часового розпаралелювання. Цей метод базується на розподіленні процесу оброблення даних між множиною автономних потоків, що забезпечують виконання однотипних операцій пошуку синхронно, за одиницю часу [4].

Встановлено, що загальний час виконання операцій над даними певного об'єму є обернено пропорційним кількості потоків, що виконують операції над цими даними. Застосування методу часового розпаралелювання стало можливим лише завдяки використанню багатозадачних програм-

них середовищ, до яких належать майже всі типи операційних систем нового покоління [4]. Метод часового розпаралелювання забезпечує зменшення тривалості оброблення даних, а також оптимізує витрати обчислювальних ресурсів процесора та оперативної пам'яті.

Постановка задачі

Запропонований у статті алгоритм фактично є спробою модифікації послідовного алгоритму, описаного в [1]. Цей алгоритм має грунтуватися на використанні множини автономних потоків, що здійснюють пошук ключів дерева в кожному з його кущів. Основною складовою цього алгоритму є потік, що виконує функції керування працюючими автономними потоками, зокрема створює екземпляри множини автономних потоків, перевіряє їхній поточний стан та ін. Згаданий вище керівний потік має назву "контролера потоків". Принцип дії алгоритму синхронізації виконання операцій пошуку ключів ієрархічного дерева є дуже близьким до алгоритму пошуку оптимального шляху в лабіринті колективом автономних потоків, а також декількох класичних алгоритмів оброблення даних, поданих у вигляді графів.

Розроблення програмного забезпечення із використанням запропонованого алгоритму передбачає використання бібліотек для роботи з потоковою багатозадачністю, що входять до складу майже всіх існуючих компіляторів мов програмування. До цих засобів насамперед належить підмножина функцій бібліотеки Win32 API для керування процесами та потоками, а також засоби бібліотеки MFC, зокрема клас CWinThread та ін. [4].

Алгоритм пошуку ключів ієрархічного дерева системного реєстру множиною автономних потоків

Алгоритм керування роботи потоків передбачає таку послідовність операцій (рис. 1). Початковою фазою роботи алгоритму є ініціалізація масивів знайдених ключів та дескрипторів працюючих автономних потоків. Аналогічно до алгоритму послідовного виконання операцій, на цьому ж етапі здійснюється збереження в масив значення заданого ключа, з якого починається процес пошуку. Надалі відбувається присвоєння початкових значень змінній-лічильнику ключів iKeyNodeIndex [2]. Крім вищезгаданого, у запропонованому алгоритмі для керування роботою керівного потоку використовують також додаткову логічну змінну bKillController, яка набуває значення стану завершення роботи контролера. На початку роботи контролера змінна bKillController набуває значення логічного "0".

Контролер потоків практично являє собою потік, що виконує функції керування процесом оброблення ієрархічних даних. Однією з його головних функцій є послідовне циклічне перебирання елементів масиву знайдених ключів. Для кожної отриманої з масиву множини знайдених ключів контролер потоків приймає рішення про створення та виконання автономних потоків, що виконують операції пошуку підключів кожного з отриманих ключів.

Перед початком виконання операції отримання з масиву поточного ключа з індексом, якому відповідає значення змінної iKeyNodeIndex, цей алгоритм блокує доступ до масиву ключів іншими працюючими на цей момент потоками. Для цього використовують засоби синхронізації потоків, зокрема набір функцій бібліотеки Win32 API, а також методи класу CCriticalSection бібліотеки MFC. Засоби синхронізації запобігають виникненню програмних конфліктів, пов'язаних із одночасним доступом контролера та множини потоків до масиву знайдених ключів [2, 4, 6].

Наступним етапом виконання запропонованого алгоритму є доступ до значення отриманого з масиву поточного ключа та отримання значення його дескриптора. Ці операції виконують за допомогою API-функції RegOpenKeyEx. Надалі перевіряють наявність помилок доступу до цього ключа. У випадку виникнення цих помилок виконують операції отримання з масиву та доступу до наступного ключа.

Отримавши доступ до поточного ключа, створюють екземпляр та запускають автономний потік, який виконує операції пошуку множини підключів.Значення отриманих підключів використовують для формування значень нових ключів у вигляді символьного рядка завдовжки 266 байт [1].

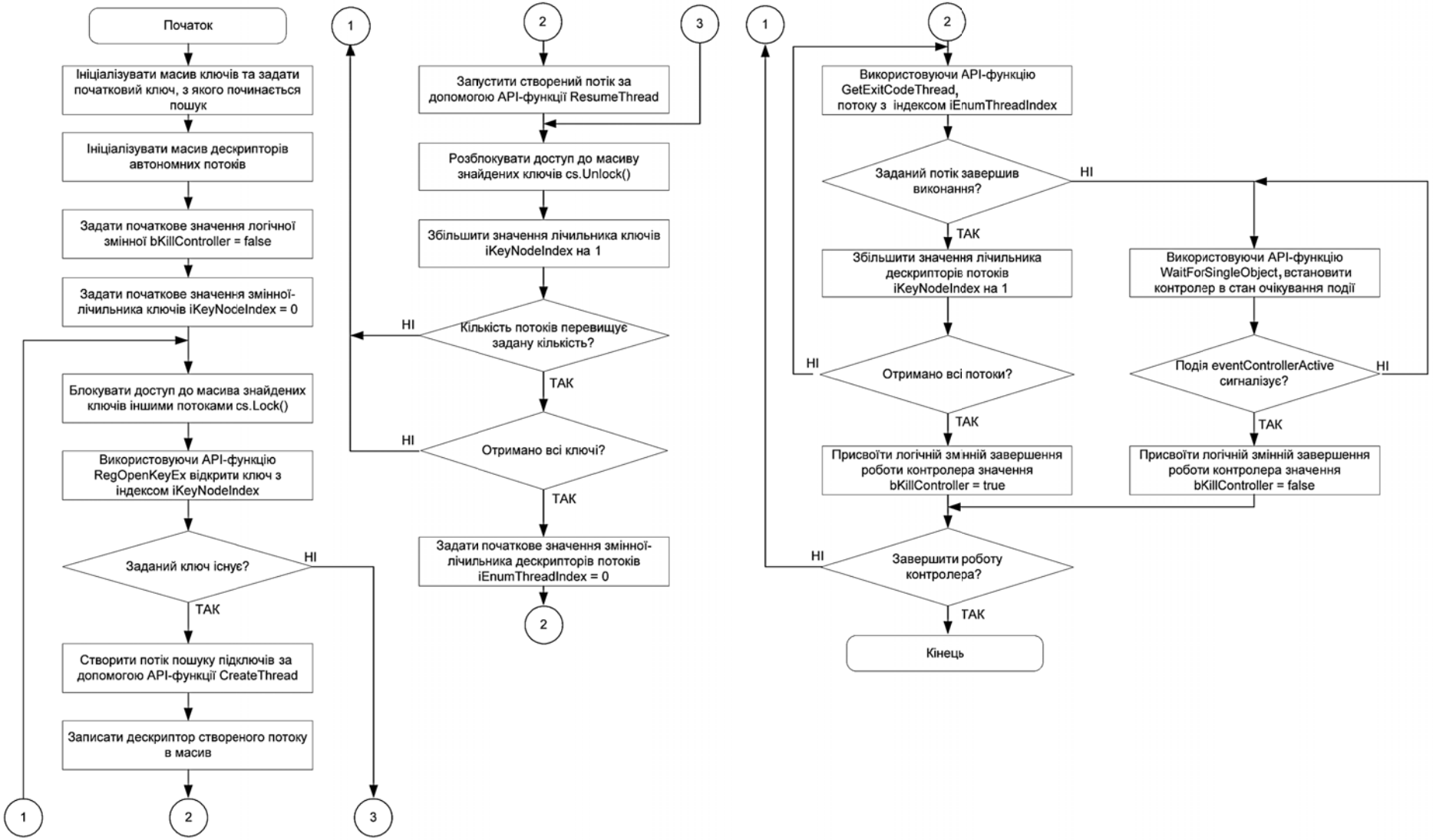

Рис. 1. Блок-схема алгоритму керування автономними потоками

Екземпляр потоку створюють за допомогою виклику API-функції CreateThread, результатом роботи якої є значення дескриптора потоку, який під час подальшого виконання операцій пошуку зберігається в масив дескрипторів. Одним з вхідних аргументів цієї функції є аргумент, значенням якого є callback-функція, тіло якої містить програмний код створеного потоку. Призначенням цього програмного коду є виконання операцій отримання під-ключів заданого ключа [2, 4, 6].

Після виконання описаних вище дій запускають потік за допомогою виклику API-функції ResumeThread, одним із значень аргументів якої є значення змінної-дескриптора створеного потоку [2, 4, 6].

Описана послідовність дій завершується виконанням операції розблокування доступу до масиву знайдених ключів засобами синхронізації згаданого вище класу CCriticalSection. Крім цього, для отримання з масиву значення наступного ключа виконують також операцію збільшення значення змінної-лічильника iKeyNodeIndex на 1.

Наступним кроком роботи контролера є перевірка допустимої кількості працюючих потоків, метою якої є усунення проблем, пов'язаних із переповненням оперативної пам'яті значень дескрипторів великою кількістю створених потоків. На цьому ж кроці перевіряють, чи під час циклічного перебирання не було отримано значень всіх існуючих в масиві ключів.

Згадані вище перевірки виконують так: якщо значення кількості працюючих потоків не перевищує заданого значення (оптимальним значенням кількості працюючих в деякий момент часу потоків є число 32), та поточне значення змінної-лічильника iKeyNodeIndex менше за величину загальної кількості елементів масиву, то повертаються до кроку, на якому отримують з масиву значення наступного ключа та виконують описані вище операції [2]. В іншому випадку переходять до виконання операції перевірки поточного стану працюючих на цей момент часу потоків. Згаданий вище процес перевірки виглядає так. Методом послідовного циклічного перебирання отримують з масиву значення дескрипторів працюючих потоків та перевіряють на предмет повернення кожним з працюючих потоків коду завершення. Якщо під час перевірки було знайдено хоча б один працюючий потік, то контролер встановлюють в стан очікування події eventControllerActive. У протилежному випадку переходять до наступного кроку виконання, на якому перевіряють отримання сигналу завершення роботи контролера. Детальний огляд виконання послідовності цих операцій наведено нижче.

Спочатку присвоюється початкове значення змінній-лічильнику дескрипторів iEnumThreadIndex=0, отримують з масиву поточного значення дескриптор, якому відповідає значення лічильника. Після цього за допомогою виклику API-функції GetExitCodeThread отримують значення коду повернення потоку. Аргумент цієї функції набуває значення отриманого з масиву дескриптора [4,6]. Результатом виконання цієї функції є значення коду повернення потоку із заданим дескриптором. Над цим значенням виконується операцій порівняння із заданим значенням коду завершення роботи потоку. Якщо ці значення є однакові, то збільшують значення змінної-лічильника iEnumThreadIndex на 1 та отримують з масиву наступне значення дескриптора.

Якщо під час виконання операцій послідовної перевірки було отримано з масиву дескрипторів значення останнього елемента, то змінній bKillController присвоюють значення логічної "1" та переходять до перевірки на предмет отримання сигналу про завершення роботи потоку.

Якщо було знайдено хоча б один працюючий потік, то викликається API-функція WaitForSingleObject, яка виконує функції очікування події eventControllerActive. Контролер продовжує виконувати операції керування тільки тоді, коли згадана вище подія знаходиться в стані „сигналізує" [4, 6]. У цьому випадку змінній bKillController присвоюють значення логічного "0" та перевіряють, чи значення цієї змінної відповідає сигналу завершення роботи контролера. Якщо змінна bKillController набуває значення логічного "0", то переходять на початок виконання алгоритму, інакше – завершують роботу контролера. На рис. 1 показано детальну блок-схему алгоритму керування роботою автономних потоків.

Охарактеризуємо алгоритм, що забезпечує отримання з реєстру множини значень підключів, що належать заданому ключу (рис. 2).

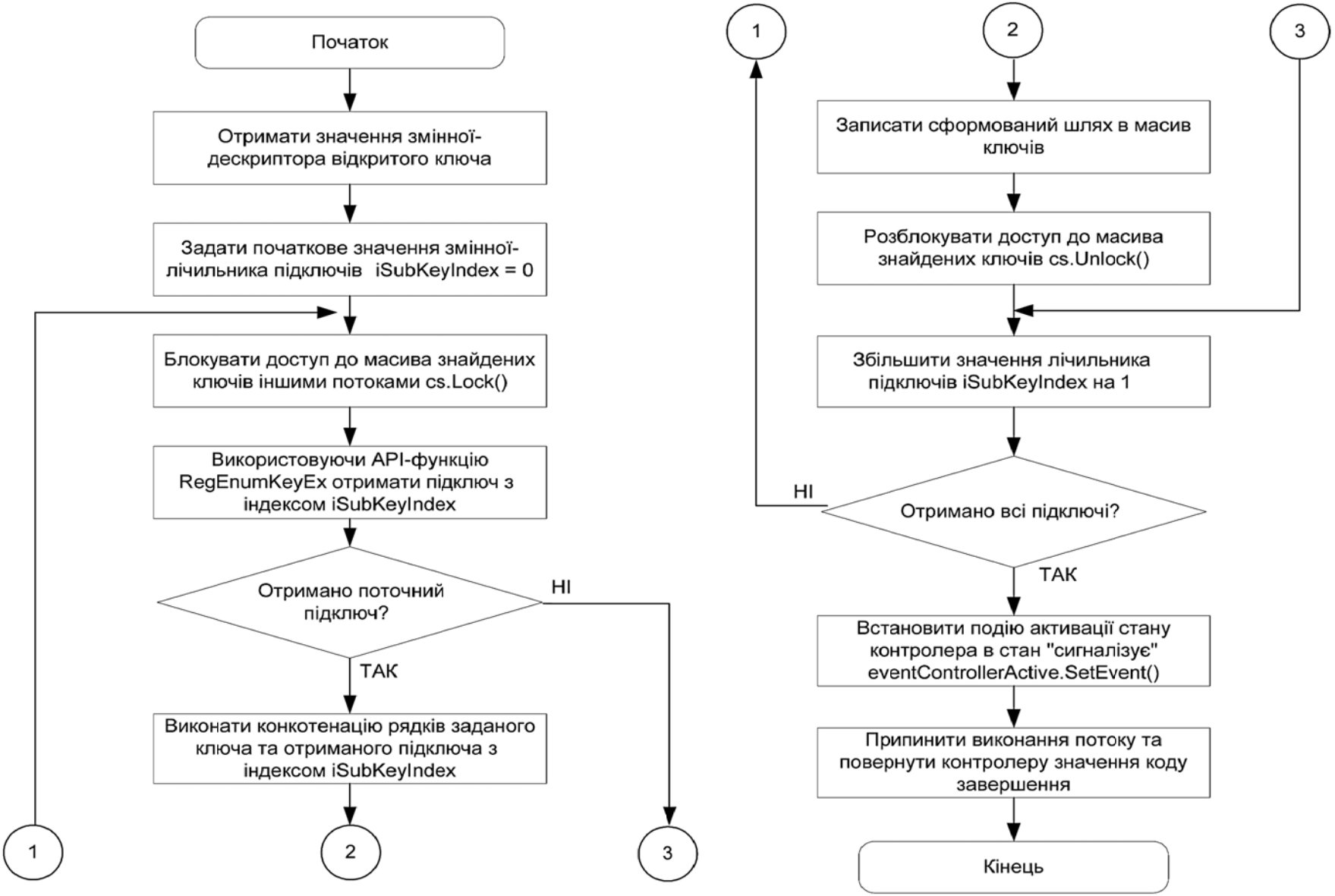

Рис. 2. Блок-схема алгоритму пошуку ключів ієрархічного дерева системного реєстру

Як було згадано вище, алгоритм пошуку підключів заданого ключа реалізовано у вигляді потоку та являє собою блок програмного коду у вигляді callback-функції, один з аргументів якої набуває значення дескриптора відкритого (під час роботи керівного потоку) ключа. Надалі це значення використовують в процесі виконання потоком функцій отримання підключів. Процес пошуку підключів заданого ключа грунтується на виконанні операцій послідовного циклічного отримання підключів з реєстру. Спочатку присвоюють початкове значення змінній-лічильнику підключів iSubKeyIndex=0, потім відбувається операція блокування доступу до масиву знайдених ключів. Наступними кроками є виклик API-функції RegEnumKeyEx, що виконує функцію отримання поточного значення підключа, якому відповідає значення змінної-лічильника iSubKeyIndex, та перевірка результату роботи цієї функції. Після цього формують значення нового знайденого ключа та зберігають його в масиві ключів. Надалі відміняють блокування масиву ключів та збільшують значення змінної-лічильника підключів iSubKeyIndex на 1.

Завершальним етапом виконання послідовності цих операцій є перевірка отримання останнього значення підключа, що належить заданому ключу. Якщо в процесі виконання операцій було знайдено останній ключ, то відбувається встановлення події eventControllerActive в стан "сигналізує". Потім відбувається передавання контролеру потоків значення коду повернення для цього потоку та завершення його виконання. Блок-схему алгоритму потоку, що здійснює пошук нових ключів ієрархічного дерева на основі заданого ключа, показано на рис. 2.

Дослідження основних показників продуктивності запропонованого алгоритму

Продуктивність запропонованого алгоритму досліджували на основі використання збалансованих ієрархічних дерев з однаковою кількістю підключів в кожному з ключів дерева. Для розрахунку основних параметрів таких дерев використовували формулу: $\sum_{i=0}^{r-1} L^i = N$, де N – кількість ключів, з яких складається дерево, L – кількість підключів ($L > 0$), що належать кожному з них, r – кількість рівнів ієрархії.

Як приклад для розрахунків основних показників розробленого алгоритму використовували дерево, в якому кількість підключів кожного існуючого ключа становить $L = 2$, за кількості рівнів ієрархії $r = 3$.

Підставивши ці значення до формули: $\sum_{i=0}^{2} L^i = 1 + L + L^2 = N$, отримуємо величину кількості ключів експериментального дерева $N = 7$.

Для оцінки ефективності запропонованого методу спочатку досліджують його вплив на процес пошуку ключів з погляду загальної кількості виконаних операцій за одиницю часу. Практика показує, що в обох випадках у разі послідовного та паралельного виконання операцій пошуку загальна кількість цих операцій не змінюється і дорівнює $p = 42$ однотипних операцій [1].

Як було згадано у попередніх розділах статті, операції пошуку ключів ієрархічного дерева у цьому алгоритмі виконує група автономних потоків. Тому відповідні обчислення загального часу виконання цих операцій розпочинаються з визначення загальної кількості груп автономних потоків t.

Для отримання значення величини t насамперед обчислюють кількість операцій p_i виконаних кожним i-м потоком, який працює в деякий момент часу. Надалі з отриманої множини $\overline{p}$ вибираємо всі ненульові значення $\forall p_i > 0$, знаходимо їхню кількість та збільшуємо це значення на 1. Скориставшись цим правилом, отримуємо: $\overline{p} = \{2,2,2,0,0,0,0\}$, $t = 3 + 1 = 4$ груп автономних потоків.

Це правило також застосовують і для обчислення кількості потоків q_k, що належать до k-ї групи. Кількість потоків у першій групі $k = 1$ завжди має становити $q_1 = 1$. Для кожної наступної

групи кількість потоків обчислюють за допомогою формули: $q_k = p_{k-1}$, $k = \overline{2,t}$. Виконавши розрахунки, отримуємо відповідні значення : $q_1 = 1$, $q_2 = p_1 = 2$, $q_3 = p_2 = 2$, $q_4 = p_3 = 2$.

Визначивши кількість груп потоків та кількість потоків в кожній групі, обчислимо час виконання операцій пошуку кожним з працюючих потоків *k*-ї групи за допомогою формули:

$$\tau_{ik} = \sum_{j=1}^{m} \left(\delta_{зчит.}^{ij} + \delta_{форм.}^{ij} \right), \tag{1}$$

де τ_{ik} – час виконання i-го потоку k-ї множини; $\delta_{зчит.}^{ij}$ – час отримання *j*-го підключа, що належить і-му ключу; $\delta_{форм.}^{ij}$ – час формування j-го ключа та запису його в масив; i – номер потоку ($i \le N$); j – номер знайденого підключа; k – номер групи потоків ($k \le t$); m – кількість знайдених підключів ($m \le L$).

Для оцінки найгіршого часу знаходження ключів заданого ієрархічного дерева (див. статтю [1]), приймаємо значення часу виконання операцій $\delta_{дост} = \delta_{зчит} = \delta_{форм} = 1$ мс.

У результаті, використовуючи формулу (1), отримуємо такі часові величини:

$\delta_{зчит.}^{ij} + \delta_{форм.}^{ij} = 1+1 = 2$,

$\tau_{11} = 2 \times L = 2 \times 2 = 4$ мс., $q_1 = 1$,

$\tau_{12} = 2 \times L = 2 \times 2 = 4$ мс., $\tau_{22} = 2 \times L = 2 \times 2 = 4$ мс., $q_2 = 2$,

$\tau_{13} = 2 \times L = 2 \times 0 = 0$ мс., $\tau_{23} = 2 \times L = 2 \times 0 = 0$ мс., $q_3 = 2$,

$\tau_{14} = 2 \times L = 2 \times 0 = 0$ мс., $\tau_{24} = 2 \times L = 2 \times 0 = 0$ мс., $q_4 = 2$.

Тепер на основі попередніх розрахунків необхідно знайти час виконання операцій кожною k-ю групою потоків δ_n^k. Відомо, що кожна k-та група потоків виконує операції пошуку синхронно в часі. Внаслідок цього сумарний час виконання операцій групою потоків дорівнюватиме часу роботи потоку з найбільшою тривалістю. Відповідні обчислення здійснюють за формулою:

$$\delta_n^k = \max\{\tau_{ik}\}, \tag{2}$$

Де δ_n^k – час виконання операцій пошуку k-ю групою потоків; τ_{ik} – час виконання i-го потоку k-ї множини; i – номер потоку ($i \le N$); k – номер групи потоків ($k \le t$).

Підставивши отримані значення до формули (2), знаходимо величину часу виконання операцій кожною з груп потоків:

$\delta_n^1 = \max\{4\} = 4$ мс., $\delta_n^2 = \max\{4,4\} = 4$ мс., $\delta_n^3 = \max\{0,0\} = 0$ мс., $\delta_n^4 = \max\{0,0\} = 0$ мс.

Розрахувавши всі необхідні величини, знайдемо сумарний час синхронного виконання операцій пошуку за формулою:

$$\delta = \sum_{i=1}^{n} \delta_{дост.}^{i} + \sum_{k=1}^{t} \delta_n^k, \tag{3}$$

де δ – загальний час пошуку ключів дерева; $\delta_{дост.}^{i}$ – час доступу до і-го знайденого ключа; δ_n^k – час виконання операцій пошуку k-ю групою потоків; i – номер отриманого з масиву ключа ($i \le N$); k – номер групи потоків ($k \le t$); t – кількість груп потоків.

Скориставшись формулою (3), отримуємо:

$$\sum_{k=1}^{4} \delta_n^k = \sum (4,4,0,0) = 8 \text{ мс.}, \qquad \sum_{i=1}^{n} \delta_{дост.}^{i} = 1 \times N, \quad \delta = 1 \times N + 8 = 7 + 8 = 15 \text{ мс.}$$

У результаті проведених вище обчислень було знайдено найгірший час пошуку ключів ієрархічного дерева у разі паралельного виконання операцій. Ця величина становить $\delta = 15$ мс.

Проведемо порівняльний аналіз цих показників за синхронного та послідовного виконання. З розрахунків, проведених в статті [1], відомо, що найгірший час послідовного виконання операцій пошуку становить $\delta_0 = 35$ мс. Величину виграшу в часі розраховують за допомогою відношення: $\delta = \frac{\delta}{\delta_0} \times 100\% = \frac{15}{35} \times 100\%$. Отже, виграш у часі виконання пошуку у випадку застосування методу часового розпаралелювання становить $\delta_в = 42{,}8\%$ за однакової кількості однотипних операцій $p = 42$.

Висновки

Застосування методу часового розпаралелювання для розроблення алгоритму пошуку ключів ієрархічного дерева системного реєстру значною мірою зменшує витрати в часі та підвищує швидкість процесу оброблення великих масивів ієрархічних даних. Сумарний виграш в часі порівняно із застосуванням методу послідовного виконання операцій пошуку становить 42,8 %. Крім високих показників часу виконання операцій, метод часового розпаралелювання також забезпечує значну економію обчислювальних ресурсів процесора та оперативної пам'яті.

На відміну від запропонованого раніше алгоритму, описаний у цій статті алгоритм передбачає застосування множини автономних потоків, що виконують операцій пошуку ключів дерева за одиницю часу. Цей алгоритм також передбачає розділення функцій пошуку ключів ієрархічного дерева окремими потоками та керування ними;

Запропонований у статті алгоритм синхронізації виконання операцій пошуку ключів ієрархічного дерева системного реєстру є достатньо простим та може застосовуватися для розроблення програмного забезпечення із застосування більшості з існуючих мов та технологій програмування.

1. Демида Б.А., Ратз А.В. Алгоритм пошуку ключів ієрархічного дерева системного реєстру WIN32 // Вісн. Нац. ун-ту "Львівська політехніка". – 2004. – № 521. 2. Кнут Д. Искусство программирования. Т. 3: Сортировка и поиск. – 2-е изд. – М., 2000. – 832 с. 3. Грегори К. Использование Visual C++ 6: Специальное издание. – М.: Изд. дом "Вильямс", 1999. – 864 с. 4. http://www.citforum.ru/ – веб-сайт Центру інформаційних технологій МГУ – "CIT ". 5. http://msdn.microsoft.com/ – веб-сайт Microsoft Developers Network Online.