УДК 681.03

А.Ю. Дорошенко, М.В. Котюк, С.С. Ніколаєв

МУЛЬТИАСЕМБЛЕРНА МОВА ПРОГРАМУВАННЯ

Запропоновано нову мультиасемблерну мову програмування високого рівня МАЯ, що поєднанує у собі властивості динамічності, крос-платформності та високої продуктивності, а також відсутність потреби в компіляторі та можливість простої апаратної інтерпретації.

Вступ

Сьогодні для створення і використання інформаційних технологій існує безліч мов програмування, що націлені для вирішення задач певного типу. Чим ширше коло задач покладається на мову, чим потужнішою є її функціональність і тим складнішими стають семантика і синтаксис мови. З іншого боку, виникнення і розвиток різних мов програмування диктується ще й розвитком особливостей платформ, для яких призначаються виконання написані цими мовами програми. Ускладнення мов програмування призводить до утруднення вивчення та програмування на цих мовах, до того ж жодну сучасну мову програмування не можна вважати дійсно універсально прийнятною і ефективною.

У зв'язку з цим виникає питання щодо створення нових кросплатформних мов програмування з компактним синтаксисом і інтуїтивно зрозумілою семантикою, програми для яких мали б високу продуктивність при виконанні. В даній роботі зроблена спроба розробити таку мову, названу МАЯ, разом з віртуальною машиною для неї. При цьому за мету поставлено знаходження балансу між компактністю та функціональністю мови і вже як наслідок – розробка синтаксису мови.

МАЯ (від російської назви «мультиассемблерный язык») – це нова мова програмування високого рівня з новим підходом до створення віртуальної машини для виконання програм, написаних цією мовою. Вона не належить до жодного з існуючих класів мов програмування і створена під впливом процедурних мов МИША [1] та Аналітик [2], підтримує основні можливості цих мов, а також емулює деякі засоби декларативних та сценарних (скриптових), таких мов як Lisp, Haskell, PHP та Java Script. Її основні властивості – поєднання динамічності, крос-платформності, високої продуктивності, а також відсутність потреби в компіляторі та можливість простої апаратної інтерпретації. МАЯ створена з використанням програмної платформи МИША (версія M10) і є гарною демонстрацією можливостей останньої для створення інструментальних засобів програмування. Водночас МАЯ є стандартним компонентом поточної версії платформи МИША, призначеним для програмування інтелектуальних алгоритмів. Також ця мова може бути використана як незалежний інструмент для проектування веб-систем та автоматизації роботи з комп'ютером (створення командних сценаріїв).

МАЯ – це мова з інтернаціональним синтаксисом, в основі якого лежать не ключові слова, а математичні позначення і авторські «ключові символи», які є замінниками ключових слів. У подальших розділах роботи міститься короткий опис засобів мови та основні методи роботи, що ілюструються прикладами і поясненнями. Особливу увагу приділено роботі зі змінними, так як вони складають чи не найбільш цінну частину розробки.

Мова МАЯ та віртуальна машина МАЯ

Спочатку мова МАЯ призначалася для швидкого виконання невеликих командних сценаріїв. Проте згодом виявилось, що мова може бути успішно використана не тільки для опису командних сценаріїв, але й для написання різних програм. Внаслідок компактності свого коду та його швидкодії, які на порядок вищі, ніж у PHP та JavaScript, мова МАЯ може

бути також використана й у Інтернет технологіях, і для задач штучного інтелекту. Концепція програмування на мові МАЯ (мікропрограмне управління) як найкраще підходить для мультипотокових систем, для яких необхідні алгоритми здатні ефективно виконуватись у паралельному середовищі (особливо, на процесорах з мультиядерною архітектурою [3]).

МАЯ – це алгоритмічно-орієнтована, імперативна крос-платформна мова програмування з підтримкою засобів об'єктно-орієнтовного програмування (ООП). При створенні мови програмування МАЯ за основу були взяті мови МИША та Аналітик. У цій мові програмування поєднуються дуже важливі якості – потужна функціональність, компактність коду та простота в освоєнні. Семантика багатьох конструкцій та елементів мови запозичена з відомих мов програмування FORTRAN, C, Pascal, Lisp, МИША. Це помітно знижує початкові зусилля при вивченні МАЯ, а підтримка ООП дозволяє створювати програми з допомогою декларативних засобів. При дотриманні принципів функціонального чи модульного програмування це дозволяє максимально ефективно оптимізувати спільну роботу в певному проекті та повністю використовувати наявні обчислювальні ресурси.

Мова МАЯ виконується на однойменній віртуальній машині. Завдяки використанню технології керування пам'яттю комп'ютера з допомогою символьних імен, мікропрограмного аналізу коду програми та тунельної індексації при виконанні, вдається значно прискорити виконання програм, написаних на мові МАЯ. Можна стверджувати, що втрати швидкості виконання відсутні, окрім втрат на виконання самої віртуальної машини, через те, що вищезгадані технології дають можливість уникати перевірок багатьох зайвих умов, одразу перевіряючи тільки найнеобхіднішу з них, і без затримок продовжувати виконання.

Віртуальна машина МАЯ працює наступним чином. Код програми передається до віртуальної машини МАЯ, де піддається попередньому аналізу. Аналіз коду виконується один раз при першому запуску програми для відстеження синтаксичних та логічних помилок. При цьому аналізуються всі синтаксичні конструкції, вирази та структура програми. При наявності помилок у коді програми, МАЯ інформує користувача про тип помилки та її місце знаходження (детальніше про це – у розділі далі). Програма не запускається на виконання, якщо у її коді наявна хоча б одна помилка.

Після того як програма проаналізована, стає можливим збереження аналізу структури її коду у вигляді масиву байтів. Даний масив являє собою область віртуальної оперативної пам'яті комп'ютера у якій містяться основні характеристики, які необхідні для виконання даної програми. Цей масив можна зберегти у файлі на жорсткому диску комп'ютера, або на іншому носієві інформації. Якщо програма була проаналізована раніше і після цього не змінювалась, то етап попереднього аналізу коду не виконується, а одразу починається виконання самої програми.

Механізм віртуальної машини МАЯ починає виконувати код після першої екрануючої послідовності (***$M***),або від самого початку програми, якщо така послідовність відсутня, і продовжує виконання до того моменту, коли зустріне екрануючу послідовність (***$***), або дійде до кінця програми. Також передбачена можливість форсованої зупинки програми на будь-якому етапі виконання.

Особливості синтаксису та семантики мови МАЯ

Засоби мови МАЯ містять команди різних рівнів, і водночас код на мові МАЯ не можна віднести до жодної категорії проміжного коду (наприклад, байт-коду Java [4], або коду MSIL [5]), до того ж жодна високорівнева мова не може бути виконана тим способом, яким виконується МАЯ. Тому в цих відношеннях мова є унікальною.

Ще однією цікавою особливістю мови є її семантика. Кожна програма розглядається віртуальною машиною МАЯ як функція, яка дуже нагадує мікропрограму [6]: 1) може отримувати параметри від інших подібних програм і запускати на ви-

конання подібні програми; 2) має власний „контейнер”, в якому можуть зберігатися довільні параметри-константи, що використовуються або можуть бути використані у даній програмі-функції; 3) має власний результат. Поняття такої мікропрограми-функції є змістом терміну *евристика,* що застосовується в мові МАЯ. Нагадаємо, що усталеним тлумаченням цього терміну є натупне:

„Евристикою можна вважати програму, програмну систему чи алгоритм, здатний до оптимізації своєї поведінки на основі розрахунку оптимальної стратегії, шляхом зміни параметрів-констант, закладених у своєму алгоритмі, на основі підбору оптимальних рішень, вхідних даних та результатів.”[1]

Кожна евристика може бути як окремою програмою, так і частиною великої системи (просто функцією). Тому, використовуючи функціональне програмування, за допомогою МАЯ можна створювати досить складні програмні комплекси. Але, завдяки „контейнеру”, кожну евристику можна також розглядати і як об’єкт з певними властивостями, які можна змінювати, що дає змогу будувати проекти у мові МАЯ за принципами об’єктно-орієнтованого програмування.

Принципи евристично-мікропрограмного програмування виявляються дуже ефективними при паралельних обчисленнях. Так, створюючи програми за цими принципами, можна отримувати програми, готові до паралельного виконання. Віртуальна машина МАЯ може використовувати відкладені обчислення подібно до декларативних мов, виконувати тунельно-індексне розпаралелення циклів, забезпечувати паралельну роботу спроектованих з допомогою МАЯ нейронно-мережевих ПС.

Розглянемо синтаксис мови. Конструкція розгалуження у МАЯ за семантикою схожа на однойменну конструкцію у мовах Паскаль та МИША.

1. Повна форма

```
?( \Умова \)
\ код, який виконується при виконанні умови\
?E
\ код, який виконується при неиконанні умови\
?^          \Кінець конструкції \
```

2. Скорочена форма

```
?( \Умова \)
\ код, який виконується при виконанні умови\
?^          \Кінець конструкції \
```

Якщо умова не виконується, виконання програми продовжується після послідовності **?^** .

Конструкція перехоплення помилок визначається за схемою:

```
$T \Початок конструкції\
\ код, який може містити помилки\
$E \перехоплення помилки\
\Реакція при виникненні помилки\
$^          \Кінець конструкції \
```

Якщо, у коді між ключовими словами ($T і $E) даної конструкції під час виконання програми буде виявлено помилку (наприклад, ділення на нуль), то програма не буде зупинена, а продовжить своє виконання після ключового слова $E даної конструкції. Це дає змогу відвернути не бажану завчасну зупинку програми, а також точніше локалізувати місце помилки та завчасно попередити її прояв.

Конструкція оголошення змінних починається послідовністю **$V** , після якої йде ім’я змінної і яка закінчується символом „**;**”.

У цій конструкції можна оголошувати скільки завгодно змінних, перераховуючи їх імена через кому. Оголошені такою дією змінні будуть видимі тільки для даної евристики.

У МАЯ є можливість створення локальних процедур, які є видимими лише для тої евристики, в якій вони оголошені. Ці процедури не мають ніяких параметрів і можуть оперувати як локальними змінними даної евристики, так і глобальними змінними усього проекту.

[1] Академия Наук СССР / „Теория распознавания образов и обучающихся систем” / Техническая кибернетика 1963 г. №5 с.98-102

```
~proc_name~^        \початок процедури\
\Код процедури\
^    \завершення та вихід із процедури\
$M\Логічний початок виконання програми\
@proc_name;         \Виклик процедури\
$.          \Логічний кінець програми\
```

Оголошуються процедури такою послідовністю: між символом **~** та послідовністю **~^** оголошується ім'я процедури (в даному випадку proc_name). Завершенням процедури вважається перший символ **^** на який натрапить віртуальна машина при виконанні даної процедури.

Для виклику процедур використовується запис **@** ім'я процедури **;** або запис **@:** ім'я змінної, або вираз, що містить ім'я процедури **;**.

Приклад:

```
~proc_name~^        \початок процедури\
\Код процедури\
^    \завершення та вихід із процедури\
$M\Логічний початок виконання програми\
$V var;
var="proc_name";\Присвоєння змінній процедури\
@:var;          \Виклик процедури\
$.          \Логічний кінець програми\
```

Такі процедури стають дуже вигідними при використанні їх у циклах.

У мові МАЯ є можливість створення різноманітних циклів. Далі будуть наведені приклади деяких з них.

Приклад програми з використанням циклу з післяумовою

```
$V var;\оголошення змінної var\
~cyc~\початок циклу\
var++;
\тіло циклу\
?(var==12)
shm{"Кінець циклу"}
?E @cyc; ?^
```

Також конструкція **~**cyc**~** **@**cyc**;** може бути використана, як вказівка goto, де конструкція **~**cyc**~** є міткою переходу з іменем „cyc", а конструкція **@**cyc**;** – вказівкою переходу до мітки з іменем „cyc".

Коментарі у мові МАЯ багаторядкові. Коментар починається та закінчується символом **\ ,** а все що поміщено в нього ігнорується віртуальною машиною.

Стиль запису виразів у мові МАЯ майже ідентичний запису виразів у мові С, але має чимало особливостей. Запис виразів відбувається у звичайній інфіксній формі. Для структурування можна використовувати круглі дужки. Наприклад: a + = 15.2· (b – c)/x; z = ((a < b)&(c > x)). Результатом виконання першого виразу буде збільшення значення змінної a на значення виразу 15.2·(b – c)/x. Результатом виконання другого виразу буде присвоєння змінній z значення істини (true), якщо і значення змінної a буде меншим від значення змінної b, і значення змінної c буде меншим від значення змінної x. Інакше змінній z буде присвоєно значення false.

Звичайно, програма не обов'язково має містити всі ці конструкції. Прикладом найпростішої програма на мові МАЯ, яка виводить на екран повідомлення «Hello from MAYA», є програма:

```
shm{" Hello from MAYA"}
```

де shm{} – функція виведення на екран повідомлення.

Далі представлений алгоритм підрахунку ряду чисел Фібоначі, реалізований мовою МАЯ.

```
\Алгоритм підрахунку чисел Фібоначі\
$V var,a,b,c;
a=1;b=1;
   ~cyc~
      ar(var)+=a;
      c=a+b;
      a=b;
      b=c;
   ?(self["Up to number"]<=Last{var})
   @cyc;
   ?^
self["Res"]=var;
```

Виконання програм та робота з даними у мові МАЯ

Виконання програм МАЯ можна зобразити наступною діаграмою (рис. 1).

При виконанні програм МАЯ у віртуальній машині (ВМ) МАЯ працює система відстеження та повідомлення про логічні і технічні помилки, наявні у коді евристик. Дану систему умовно можна поділити на дві частини, перша з яких повідомляє про синтаксичні помилки (наприклад, незавершені конструкції і подібне), які відстежуються на етапі попереднього аналізу при першому запуску евристики після її модифікації, а друга – безпосередньо відстежує логічні помилки під час виконання (наприклад, ділення на нуль). Є також можливість одразу виводити повідомлення про помилку після того як вона буде знайдена.

У прикладі повідомлення, наведеному далі, повідомляється про три синтаксичні помилки: 1) невідома конструкція $" (у рядку 25); 2) конструкція ?(Condition) не завершена (відсутній символ ?^); 3) незакритий коментар (у рядку 36); а також повідомлення про те, що програма містить грубі помилки і не може бути виконана.

```
<Syntax_Error
VMAsmCnt='"25"' Unknown_command=
="" Construction='$'>
</Syntax_Error>

<Syntax_Error
Construction='If-Then(?(Condition))'
SubConstruction='?^'> Not_Finished
</Syntax_Error>

<Syntax_Error VMAsmCnt='36'
Construction='\'> No_Match </Syntax_Error>
<Log_Error VMAsmCnt='52'>
Compiling.Failed </Log_Error>
```

Також є можливість перегляду та збереження усієї структури виконання будь-якої евристики. Дана структура містить вміст усіх основних стеків віртуальної машини МАЯ на кожній ітерації виконання програми, що значно полегшує пошук та виправлення помилок. Наприклад, структура виконання евристики shm{"Hello!"} (рис. 2).

Мова програмування МАЯ використовує найбільш зручний та ефективний метод обробки даних з-поміж усіх засобів, доступних в рамках програмної платформи МИША.

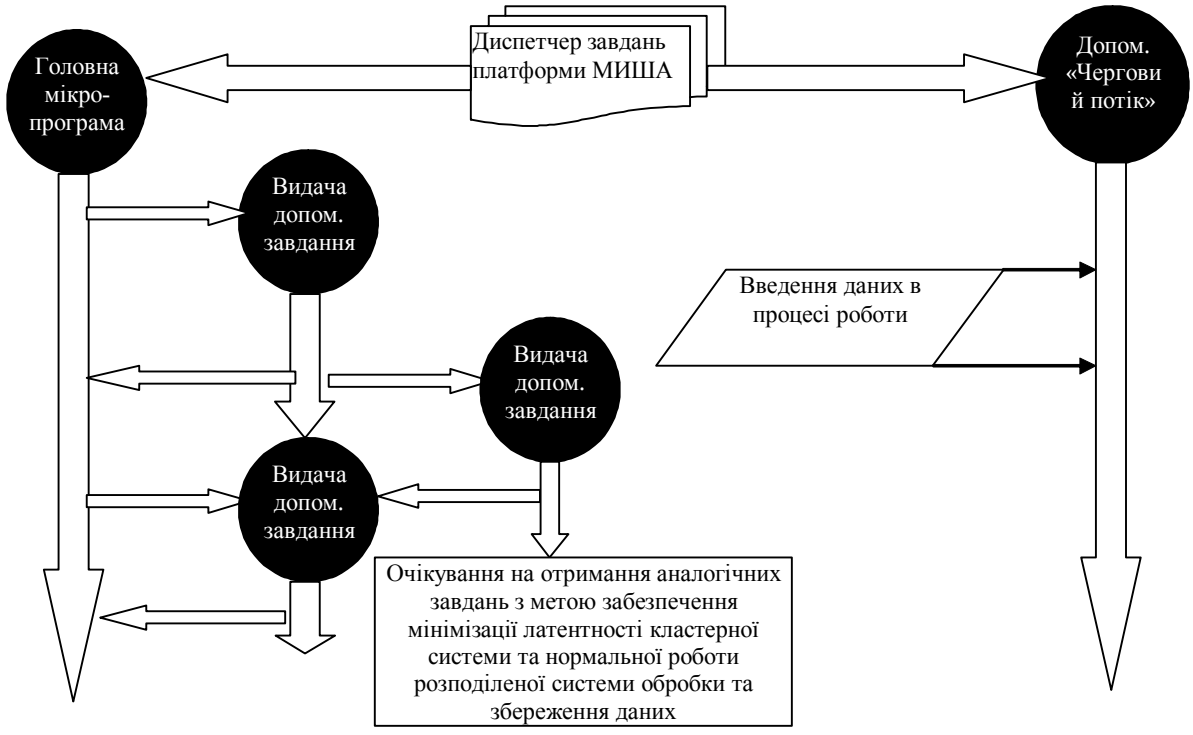


Рис. 1. Діаграма виконання програм МАЯ

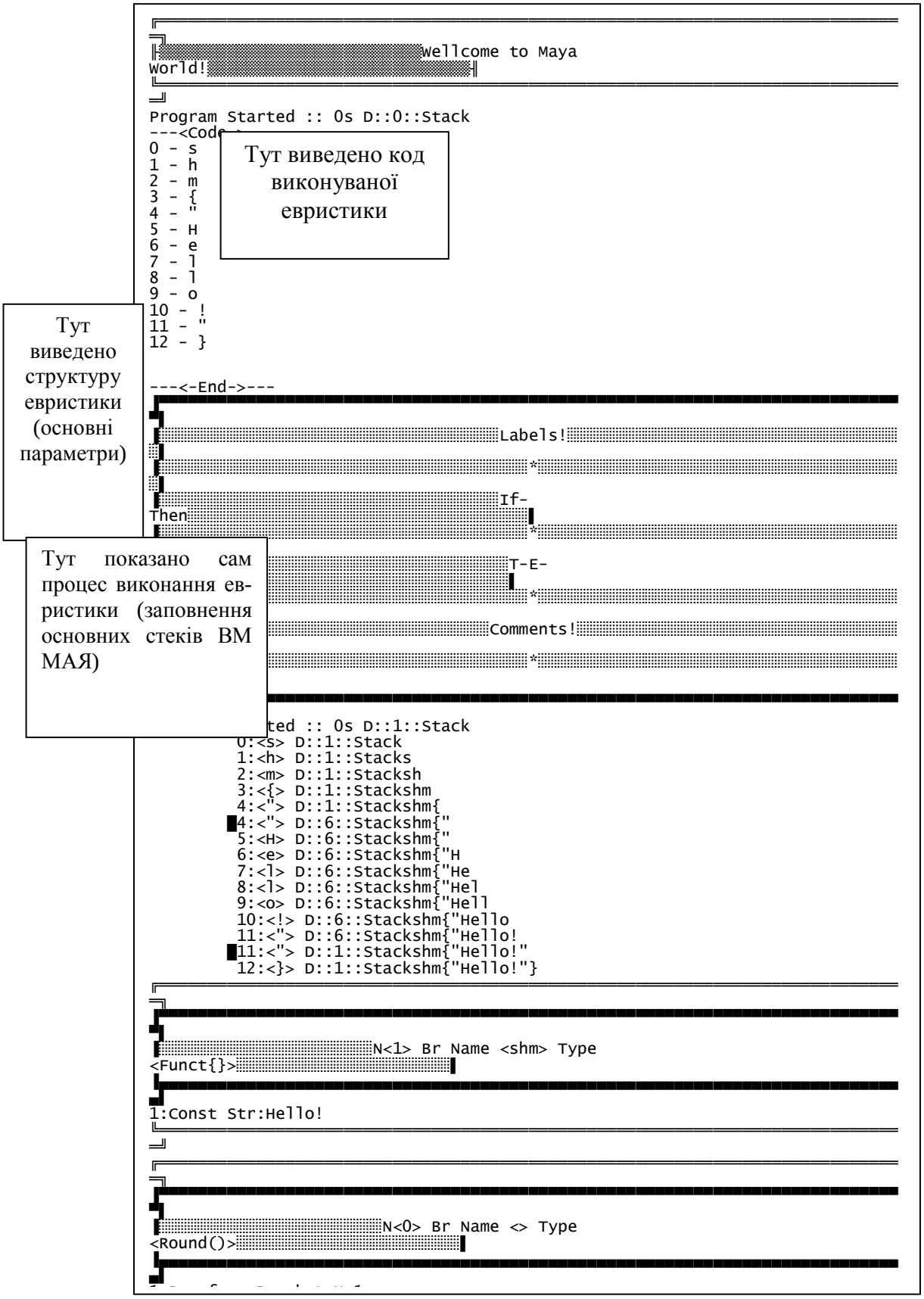


Рис. 2. Структура виконання евристики shm{"Hello!"}

На відміну від багатьох мов програмування зі статичною типізацією, у мові МАЯ немає необхідності явного визначення типу змінних, хоча така можливість існує. У разі звернення до змінної, ядро МАЯ трактує її тип відповідно до

контексту. За необхідності можливе приведення змінної до певного типу за допомогою відповідних функцій мови. Це може знадобитись, якщо зважити, що значення змінної можуть трактуватись по-різному в залежності від її типу. Так, наприклад, значення 255 може трактуватись як число „255”, або ж 255 символ таблиці ANSI – „я”. Також можливе визначення типу відповідної змінної на певному етапі виконання програми. Імена змінних можуть містити усі символи кодової таблиці, крім тих, які кодують команди мови, що може значно зменшити обсяг програм, особливо при їх автоматичному генеруванні в інтелектуальних системах (відповідно, зникає необхідність в присвоєнні змінним осмислених імен). Найбільш схожі за принципами роботи до змінних мови МАЯ атоми мови Lisp, хоча і тут є суттєві відмінності. Як і в мові Lisp, змінні МАЯ не мають явного фіксованого типу, вони можуть містити будь-що, починаючи від простих елементарних констант, закінчуючи багатовимірними масивами та надскладними структурами даних.

На відміну від багатьох інших мов програмування із змінними мови МАЯ можна утворити будь-яку структуру даних, тому автоматично відпадає необхідність у таких оголошеннях, як record у мові Pascal. Усе це робить змінні мови універсальними та унікальними. Далі наведено приклад оголошення змінної var та заповнення її структурою даних, що має вигляд масиву, у кожній чарунці якого зберігаються певні дані.

```
$V var;\оголошення змінної var\
var[0,"№"]=8123;
var[0,"ПІБ"]="Д.К. Андрійович";
var[0,"Тел"]=5468212;
var[0,"Бали"]=12;
        var[1,"№"]=5131;
var[1,"ПІБ"]="А.І. Олексіївна";
var[1,"Тел"]=5198723;
var[1,"Бали"]=8;
var[2,"№"]=1422;
var[2,"ПІБ"]="О.В. Миколайович";
var[2,"Тел"]=2894537;
var[2,"Бали"]=10;
```

Вимога однотипності наповнення масивів відсутня. Кожна чарунка може містити довільні дані, незалежно від того, що знаходиться в сусідніх чарунках. Але в даному випадку усі три чарунки однотипні. Цю структуру можна доповнювати новими чарунками, а також змінювати старі.

Слід зазначити, що не існує жодних обмежень при роботі зі змінними, крім обсягу фізичної пам'яті комп’ютера та фантазії програміста.

Технологія управління динамічною пам'яттю

Платформа МИША, на основі якої базована мова МАЯ, працює з пам’яттю у режимі інтелектуального керування на основі евристичних технологій. Запропонований підхід передбачає значно вищі показники продуктивності програм та значно нижчі показники використання ОЗП. Особливо помітні ці переваги при порівнянні із поширеним підходом на основі «збирання сміття» та технологією динамічно-стекової типізації у мовах запису командних сценаріїв.

Ознайомлення з вихідними кодами системи Mono [7], що є варіантом реалізації .NET з відкритим кодом (open source) дає змогу визначити, що алгоритм “збирача сміття” на основі побудови об'єктних графів має експоненційну складність у залежності від кількості об'єктів (N) для обробки “збирачем сміття” – $O(A^N)$, де A (кількість команд для обробки одного об'єкта). Водночас інтелектуальна технологія МИШІ дає змогу забезпечити лінійну залежність O(N) обсягу команд від кількості об'єктів. При цьому за певних значеннь N інтелектуальний алгоритм забезпечує відставання від традиційного підходу, проте при зростанні N евристична технологія забезпечує високу швидкість роботи навіть у випадку, коли “збирач сміття” теоретично не здатний обробити потрібну кількість об'єктів. Дану залежність видно на графіку (рис. 3), де по осі Ох відкладено кількість об'єктів, а по осі Оу – обсяг команд для їх обробки.

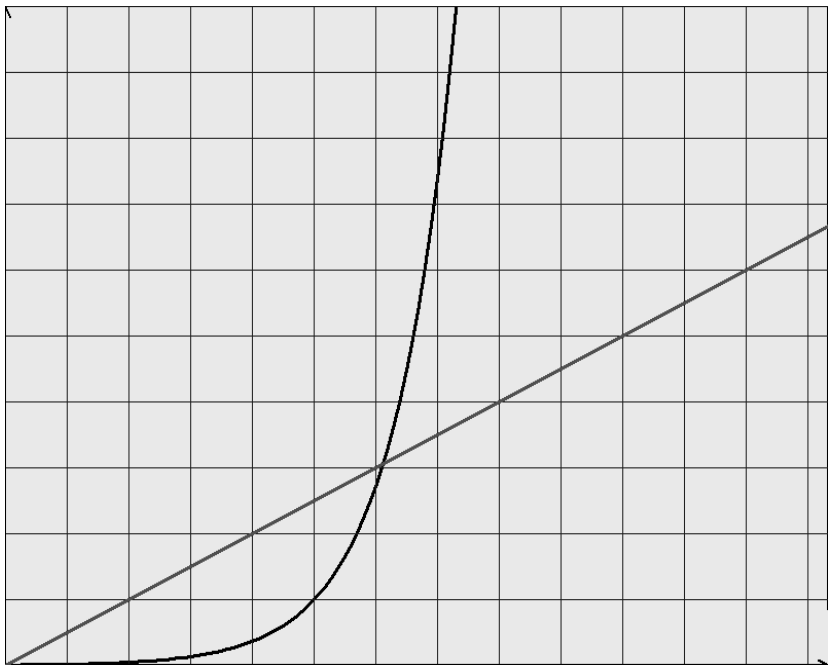


Рис. 3. Залежність складності операцій збирання сміття від кількості об’єктів

Враховуючи специфіку ситуації, в МИШІ передбачено 4 комплекси керування пам'яттю для різних типів задач, що розв'язуються з допомогою платформи:

1) виділення пам'яті в стеку для обробки даних за значенням;

2) виділення пам'яті в “купі” з використанням обліку посилань;

3) “збирання сміття” для програм, що обробляють невеликі обсяги даних;

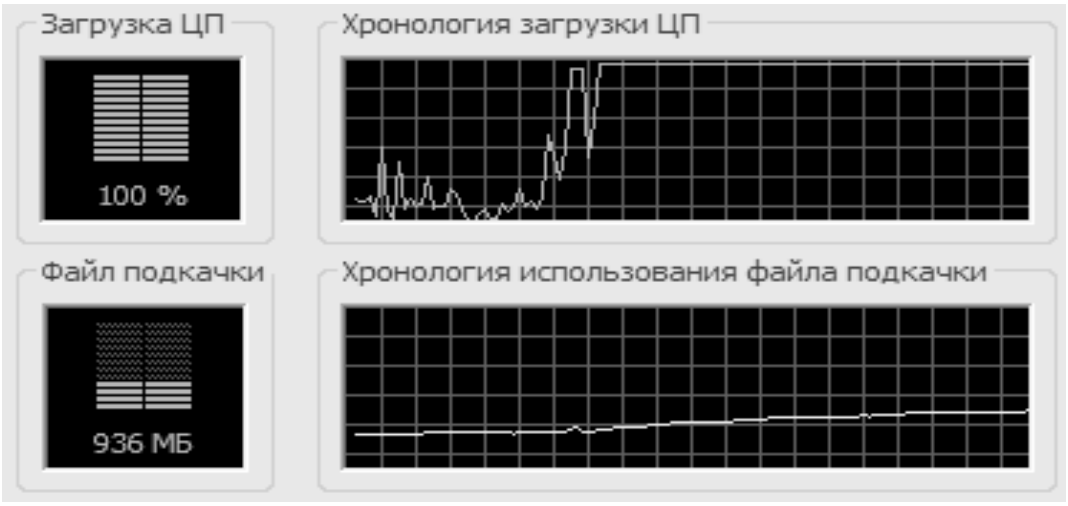
4) інтелектуальна евристична технологія керування пам'яттю в “купі”;

5) інтелектуальний альтернативний “файл підкачки” на жорсткий диск.

Для роботи з динамічною пам'яттю платформа МИША підтримує тип даних ref, який забезпечує використання динамічної області пам'яті («купи»). Використання цього типу дає змогу чітко регламентувати моменти, в які може відбутися вивільнення пам'яті – це вихід з блоку змінних-посилань та присвоєння їм нових значень. Під час виконання програми під керуванням відлагоджувача інтелектуальний аналізатор пам'яті МИШІ, реалізований як нейронна мережа, визначає ті фрагменти алгоритму, де найчастіше відбувається вивільнення даних. Усі отримані результати записуються в спеціальну базу у вигляді мікропрограм, які використовуються в подальшій роботі. Після цього у даних фрагментах виконується встановлення команди активації очистки з допомогою обліку посилань або додавання даного об'єкта в чергу очистки.

На даний момент МИША містить базу мікропрограм для керування пам'яттю обсягом близько 56 Мб, вона безперервно поповнюється під час експлуатації платформи. Проведені експерименти на алгоритмах різного рівня складності однозначно зазначають: даний підхід забезпечує економію пам'яті в обсязі, пропорційному до обсягу її гіпотетичного використання.

Розглянемо використання пам'яті та обчислювальних ресурсів у експериментальній пошуковій системі при індексації тексту з синтаксично-семантичним аналізом та порівняймо його з аналогічними показниками подібного алгоритму, реалізованого мовою програмування C# на платформі .NET 3.0. Цей алгоритм забезпечує необхідні параметри для оцінювання методів керування пам'яттю: він збільшує використання оперативної пам’яті і кількість керованих (managed) вказівників та об'єктів першого покоління, оскільки утворює складні структури-графи (рис. 4, а і б).



а

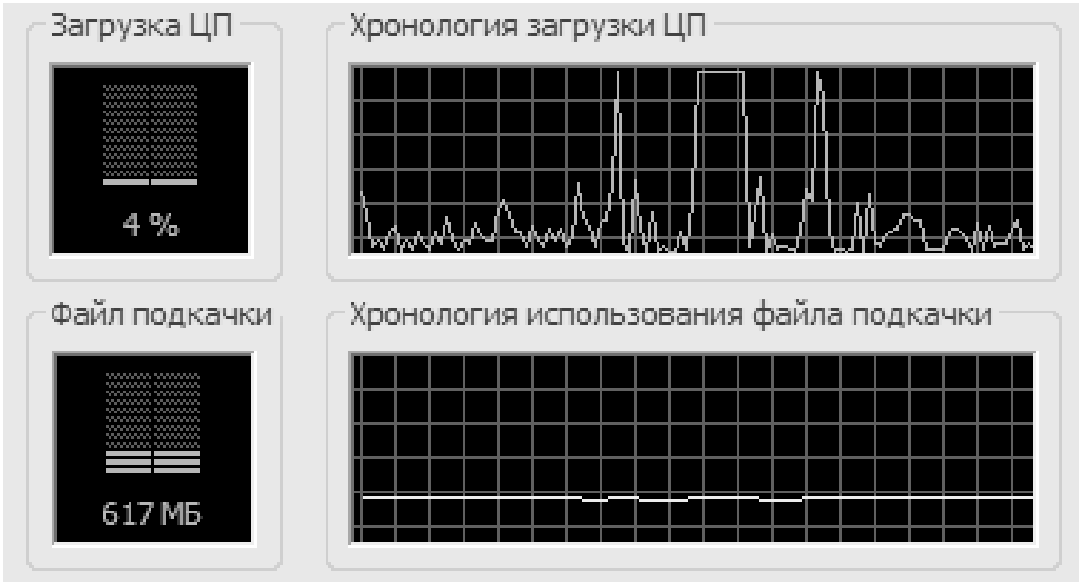
б

Рис. 4. Загальне використання ресурсів при застосуванні .NET 3.0 (а); використання виділених даному процесу квантів процесора на виконання коду MSIL (б)

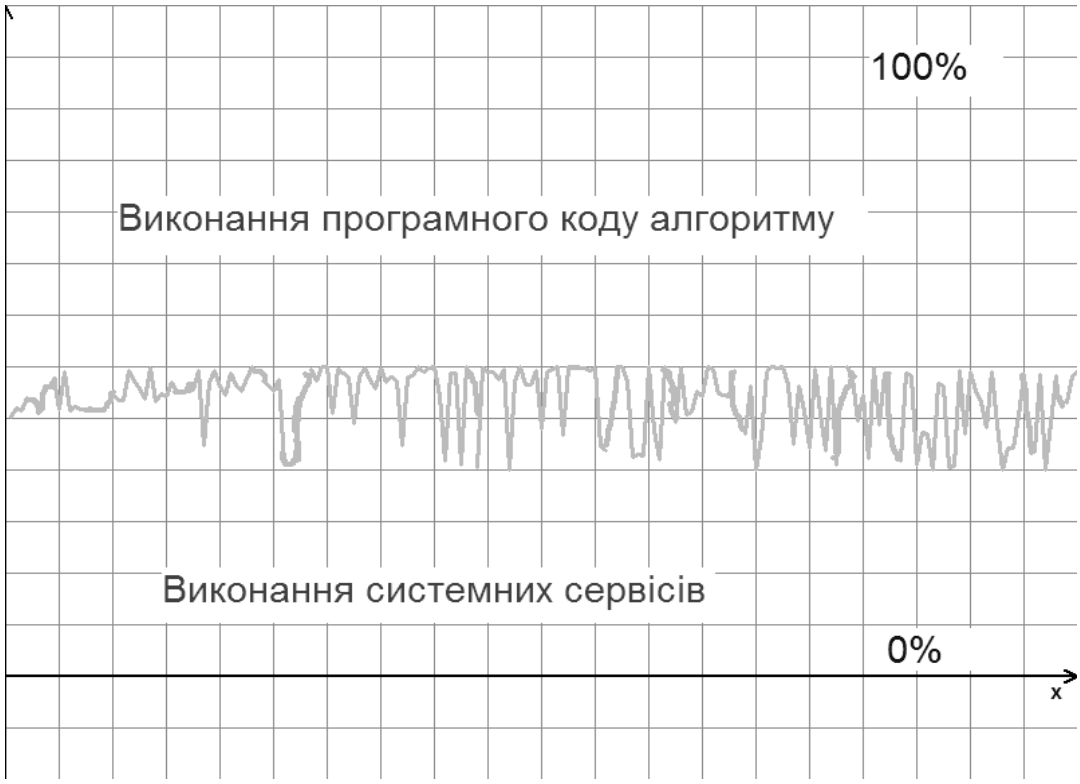
У багатопотоковому режимі виконання (на процесорі AMD Athlon 64 X2 4800+) обсяг використання пам'яті на платформі .NET залишається сталим. Використання режиму Concurrent GC дозволяє збільшити ефективність роботи програми на двоядерному процесорів, проте це не дає повного розв'язання проблеми: ми не можемо повністю використати обидва ядра процесора, а з часом відбувається перевантаження ядра, і весь процесорний час витрачається на виконання команд “збирача сміття”.

Отже, можна говорити про те, що в складних алгоритмах з обробкою великої кількості даних збирач сміття платформи .NET “захлинається” і нездатний забезпечити роботу системи з потрібними характеристиками швидкодії при необхідності керування більш ніж 50 тисяч об'єктів. Це накладає суворі обмеження на програмування складних ресурсоємних алгоритмів з допомогою платформи .NET.

Водночас, технології, використані в платформі МИША і системі керування даними мультиасемблерної мови МАЯ, дають змогу оптимізувати використання пам'яті (рис. 5, а і б).



а



Б

Рис. 5. Використання ресурсів при застосуванні МИША/МАЯ (а);
використання виділених даному процесу квантів процесора на виконання коду алгоритму
в системі МИША/МАЯ (б)

Висновки

У даній роботі запропоновано нову мультиасемблерну мову програмування МАЯ, призначену для створення швидкісних інтелектуальних алгоритмів. Підхід, запропонований для виконання программ МАЯ, забезпечує можливість реазіації цієї мови на багатьох класах обчислювачів – від мобільних пристроїв, де пам'ять є критичним ресурсом до кластерних систем, що обробляють великі обсяги даних і відеографічних прискорювачів, де на один процесор припадає всього 3,2–6,4 Мб пам'яті.

1. *Дорошенко А.Ю., Котюк М.В., Ніколаєв С.С.*. Програмна платформа для наукових досліджень // Проблеми програмування. – 2007. – № 4. – С. 49–59.
2. *Глушков В.М., Молчанов И.Н., Визнюк Г.И. и др.* «Программное обеспечение ЭВМ МИР-1 и МИР-2». – Киев: Наук. думка, 1976.
3. *Akhter S., Roberts J.*. Multi-Core Programming. Increasing Performance through Software Multi-threading. – Intel Press, 2006. – 336 p.
4. *Java 2.* Patric Haughton, Herbert Shield – Sun Developers' Guide Press, San Diego, 2004.
5. *Troelsen A.*, The C# 2005 programming language and .NET 2.0. platform. Apress, 2006.
6. *Глушков В.М., Михновский С.Д., Рабинович З.Л.* ЭВМ со структурной реализацией языка высокого уровня // Кибернетика, 1981. – № 4. С. 9 – 17.
7. *Система Mono.* http://www.mono-project.com/Main_Page

Отримано 17.12.2007

Про авторів:

Дорошенко Анатолій Юхимович,

доктор фізико-математичних наук, професор, завідувач відділу Інституту програмних систем НАН України

Котюк Микола Васильович,

Ніколаєв Сергій Сергійович,

члени Київського територіального відділення Малої академії наук «Дослідник».

Місце роботи авторів:

Інститут програмних систем НАН України.
Тел.: (38044) 526 1538,
e-mail: dor@isofts.kiev.ua

м. Київ, вул. Січневого повстання 13.
e-mail: km@p5com.com