УДК 004.415.5
С.Я. П'яла
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ДО ПРОБЛЕМИ СТВОРЕННЯ КРОСПЛАТФОРМНИХ ДОДАТКІВ В СЕРЕДОВИЩІ .NET FRAMEWORK

S.Y. Piala

ABOUT THE PROBLEM OF CROSSPLATFORM APPLICATIONS IN .NET FRAMEWORK

Для терміну «обчислювальна платформа» в контексті розробки програмного забезпечення (ПЗ) існують розширені визначення, але в рамках даної роботи ми використаємо спрощене визначення – сукупність апаратного забезпечення та операційної системи (ОС) [1].

Двохкомпонентність обчислювальної платформи є принциповим моментом, оскільки тотожність або відмінність платформи визначається кожним компонентом незалежно. Апаратне забезпечення визначається центральним процесором (ЦП) та його належністю до певного сімейства, що, в свою чергу, визначається сумісністю ЦП з множиною машинних команд. Характеристикою ОС, яка дозволяє визначати тотожність або відмінність платформ, є сумісність бінарного програмного інтерфейсу, тобто, здатності запускати на виконання програмні модулі без використання спеціальних емуляторів [2].

Проілюструємо вищесказане прикладами. Так, ЦП сімейства Intel Core або AMD Athlon при наявності тієї самої ОС Windows утворюють тотожну платформу, оскільки вони зберігають сумісність з базовою системою команд процесору i386. В той же час ЦП сімейства Intel Itanium з тією самою ОС Windows утворюють відмінну платформу, оскільки цей ЦП несумісний з множиною команд, утворених на базі i386. Один і той же процесор AMD Phenom під керуванням ОС Windows та ОС GNU/Linux утворює відмінну платформу, оскільки кожна з цих ОС не має штатних засобів виконання програмних модулів, створених для іншої ОС. А коли такі засоби існують (наприклад, підсистема POSIX в складі ОС Windows або проект WINE), вони є емуляторами і на сьогоднішній день не забезпечують повної сумісності [3].

Визначивши обчислювальну платформу, можна перейти до визначення «кросплатформності» – спеціальної властивості вихідного програмного коду, яка дає можливість генерувати програмні модулі (виконуваний машинний код) для відмінних обчислювальних платформ при відсутності або мінімальних автоматизованих змінах до вихідного програмного коду.

Потрібно чітко відрізняти поняття кросплатформності як властивості вихідного програмного коду і поняття рівня забезпечення кросплатформності – властивості компілятора мови програмування, хоча в популярній літературі ці поняття позначаються одним і тим же терміном «кросплатформність», подекуди відносячись до мови, а не до її компілятора.

Прийнято вважати, що мінімально необхідний рівень забезпечення кросплатформності для певної мови програмування досягається тоді, коли компілятор для цієї мови має змогу генерувати програмні модулі в рідному для цільової платформи форматі. Відповідно до двохкомпонентної природи платформи, така спорідненість повинна включати, по-перше, здатність генерувати машинний код для цільового ЦП, а, по-друге, відповідність структури виконуваних модулів або бібліотек внутрішнім стандартам цільової ОС, тобто відповідність так званому ABI (Application Binary

Interface). Наприклад, кросплатформний компілятор може вважатися таким, що підтримує платформу Windows x86 якщо машинний код згенерованого модуля відповідає, як мінімум, множині команд i386, а структура згенерованого модуля відповідає ABI для PE- формату (Portable Executable), який є стандартним в середовищі ОС Windows.

В стандартах одних з найбільш поширених у недалекому минулому мов програмування C та C++ закладено рівень абстракції від обчислювальної платформи у вигляді так званої стандартної бібліотеки. Заголовочні файли для стандартної бібліотеки повинні бути присутні в будь-якому середовищі розробки для цих мов, а сама стандартна бібліотека повинна бути присутня в середовищі виконання або як штатна компонента, або як супровідна бібліотека в складі ПЗ [4].

При умові, що компілятори притримуються одного і того ж діалекту мови програмування а також при умові, що програмний код не використовує виклики функцій ОС або інших бібліотек безпосередньо, це, теоретично, дозволяє проводити компіляцію вихідного коду під різні цільові обчислювальні платформи без змін. На практиці, однак, задачі, які можуть бути повністю вирішені програмним кодом в такому стилі, мають суто академічний характер і трапляються дуже рідко. Це може бути зумовлено необхідністю надання користувацького інтерфейсу або встановлення системних служб, відмінністю роботи файлових систем, навіть відмінністю реалізації стандартних бібліотек від різних вендорів тощо. З огляду на практичні міркування функціональність стандартної бібліотеки обмежена задачами високого ступеню абстракції, наприклад – керування пам’яттю, робота з файлами, масивами та векторами тощо.

Як правило, середовище виконання надає можливість користуватися програмними інтерфейсами операційної системи (або віртуальної машини) а також завантажувати додаткові програмні бібліотеки. Таким чином реалізована можливість користування додатковими бібліотеками, в тому числі і тими, які надаються сторонніми вендорами. Додаткові бібліотеки зазвичай мають вузьку спеціалізацію порівняно зі стандартною, наприклад – набір криптографічних функцій, спеціальна обробка звуку, завантаження файлів з мережі, парсинг XML-файлів тощо [4].

Для різних операційних систем можуть існувати як дистрибутиви одних і тих же додаткових бібліотек, так і різні бібліотеки зі схожою функціональністю. З точки зору ефективності виконуваного коду суттєва різниця між цими двома підходами практично відсутня.

Література

1. Порєв Г.В. Дослідження методів розробки кросплатформного програмного забезпечення, НТУУ «КПІ», 2010.

2. Дмитрий Бушенко. Программирование на .NET в Linux [Электронный ресурс] / Дмитрий Бушенко // Компьютерная газета. – 2006. – № 4. – Режим доступу: http://www.nestor.minsk.by/kg/2006/04/kg60411.html, вересень 2017 р.

3. Mark Mamone. Practical Mono – Apress, 2005 – 424 с.

4. Эндрю Троелсен. Язык программирования C# 2010 и платформа .NET 4.0 / Эндрю Троелсен; пер. с англ. В. Юрмин. – [5-е изд.]. – М.: Издательский дом «Вильямс», 2010. – 1392 с.