УДК 004.002
Мельник В.М., Багнюк Н.В., Мельник К.В., Жигаревич О.К.
Луцький національний технічний університет

РЕАЛІЗАЦІЯ С- ТА JAVA-ІНТЕРФЕЙСІВ ДЛЯ АСИНХРОННОГО РЕЖИМУ ПЕРЕДАЧІ ДАНИХ

В.М. Мельник, Н.В. Багнюк, К.В. Мельник, О.К. Жигаревич. Реалізація С- та JAVA-інтерфейсів для асинхронного режиму передачі даних. Розглянута технологія передачі даних, що реалізована з застосуванням асинхронного режиму передачі (АРП), яка пропонує власні можливості для користувачів. Для використання переваги АРП з одночасним поєднанням його роботи в прикладних програмах необхідно використовувати потужний програмний інтерфейс для прикладних додатків (ПІПД). В роботі реалізовано три різні варіанти програмного інтерфейсу для прикладних додатків з використанням вільнодоступних власних АРП-функцій. Представлені також дизайн та реалізація ПІПД для АРП на мові С для робочих станцій Digital Alpha, що працюють з системою Digital UNIX. Описано і ПІПД для АРП мовою Java, який пропонує програмування Java-сокетів в середовищі, звичному для Java-програмістів. З метою виявлення пропускних характеристик були співставлені різні концепції та архітектурні підходи
Ключові слова: Технологія передачі, асинхронний режим передачі, програмний інтерфейс для прикладних додатків, сокети, архітектура.

V.M. Melnyk, N.V. Bahnyuk, K.V. Melnyk, O.K. Zhyharevych. C- and java- interface implementation in asynchronous data transfer mode. Flexible transmission technology is represented through an asynchronous transferring mode (ATM), which proposals new services set for users. In aim to satisfy ATM's strength advantage and to integrate access with ATM services into particular applications, it need to use a powerful application programming interface (API). Three different variants of such API was implemented here with native ATM functions accessing. It is also presented ATM API design and implementation in C for Digital Alpha workstations running under Digital UNIX system. There also Java ATM API is described with offering the socket programming environment, suitable for Java-programmers. The different attitudes and architectural approaches are compared in order to give throughput information
Keywords: transmission technology, asynchronous transferring mode, application programming interface, sockets, architectural approaches

В.М. Мельник, Н.В. Багнюк, К.В. Мельник, О.К. Жигаревич. Реализация С- и JAVA-интерфейсов для асинхронного режима передачи данных. Рассмотрена технология передачи данных, реализована с применением асинхронного режима передачи (АРП), которая предлагает свои возможности для пользователей. Для использования преимущества АРП с одновременным сочетанием его работы в программах необходимо использовать мощный программный интерфейс для прикладных приложений (ПИПД). В работе реализовано три различных варианта программного интерфейса для приложений с использованием доступных собственных АРП-функций. Представлены также дизайн и реализация ПИПД для АРП на языке С для рабочих станций Digital Alpha, работающих с системой Digital UNIX. Описаны и ПИПД для АРП на языке Java, который предлагает программирование Java-сокетов в среде, привычной для Java-программистов. С целью определения пропускных характеристик были сопоставлены различные концепции и архитектурные подходы
Ключевые слова: Технология передачи, асинхронный режим передачи, программный интерфейс для прикладных приложений, сокеты, архитектура

Вступ

В останні роки комунікаційні технології різко повернули до впровадження опто-волоконних мереж і вдосконалення технологій передачі даних, особливо таких як SONET/SDH. На базі подібних розробок високошвидкісні мережі стали реальністю, зокрема, на основі технології *асинхронного режиму передачі* (АРП) даних для Broadband-ISDN. Однак, зовсім нові можливості, які запропоновані АРП, поглинаються традиційними протоколами, що функціонують між додатками і мережевими службами. Наприклад, при передачі даних з використанням IP через АРП [1] або LAN-емуляцію [2] спостерігається спадання асинхронного режиму передачі до рівня традиційного протоколу канального зв'язку і приховування нової функціональності. Єдина можливість для додатка скористатися послугами, наданими АРП, – це використання власного АРП-інтерфейсу, який надає додаткам прямий доступ до функціональності рівнів адаптації при асинхронному режиму передачі.

Форум АРП визначив семантичний опис такого АРП-інтерфейсу для прикладного програмування (АРП ПІПД) [3,4], який було реалізовано на декількох архітектурах, таких як UNIX-похідних машинах, Linux [5] і для комп'ютерів під управлінням MS-DOS або операційної системи Plan 9 та передуючих адаптерів [6]. Останній підхід був портований на FreeBSD і [7] також має відношення до розробки, реалізації операційної системи та сигнальної підтримки споріднених додатків [7], що є прикладом транспортного обслуговування, запропонованого на вершині рівня адаптації асинхронного режиму передачі AAL-5.

В роботі описуються способи проектування та дослідження власної ППІД АРП для робочих станцій, що працюють під управлінням Digital Alpha Digital UNIX. Подається Java-реалізація для ППІД АРП по лінії рекомендацій та матеріалах АРП-формуму на її вершині для даного асинхронного режиму.

Мова С для ППІД АРП на Digital UNIX

Перший програмний інтерфейс для прикладних додатків АРП, представлений в цій роботі, був розроблений на Digital UNIX (раніше OSF/I) на робочих станціях Digital Alpha мовою С [8]. Для Digital UNIX версій 3.2 і 4.0 підсистема ППІД фіксується навколо модуля управління з'єднанням (МУЗ) в якості основного модуля (рис. 1). Стосовно трьох різних класів модулів ця архітектура встановлює драйвери пристроїв, модулі сигналізації і модулі конвергенції. В залежності від потреб додатків, модулі кожного класу можуть бути додані до МУЗ, забезпечуючи додаткову функціональність.

Як правило, драйвери пристроїв походять з методів передачі поданих даних, сформованих середовищем. Для підтримки кожного сигнального протоколу модуль сигналізації зареєстрований в МУЗ. Сигнальні канали обробляють МУЗ як регулярні (що переключаються або постійні) віртуальні канали. Конвергенція модулів, нарешті, служить для того, щоб подолати додатками можливий розрив між МУЗ та іншими модулями в просторі користувача. Реалізація IP-через-АРП, який поставляється з МУЗ, є прикладом такого модуля збіжності.

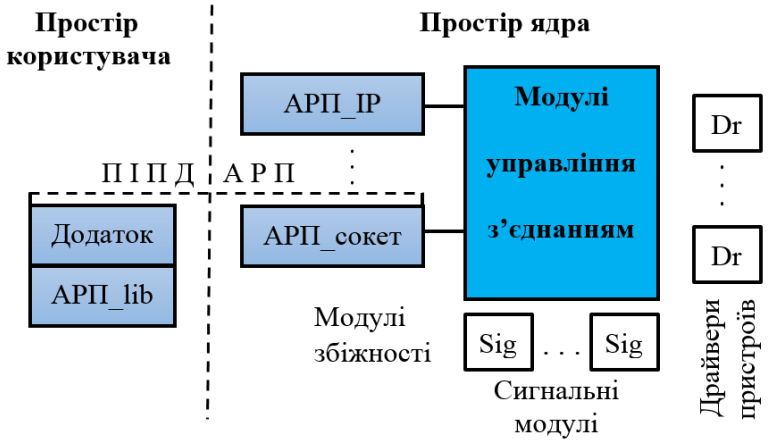

Рис. 1 – Структура підсистеми АРП digital UNIX

ППІД АРП було реалізовано також у вигляді модуля збіжності. Якщо говорити в деталях, то ППІД АРП складається з двох частин: нового модуля конвергенції atmsocket, розташованого в просторі ядра, і бібліотеки libatm в просторі користувача. Atmsocket взаємодіє з центральним МУЗ в ядрі і виконує такі завдання, як обробка контекстів зв'язку і переадресації даних користувача. Додатки отримують доступ до функцій, що забезпечують модуль atmsocket за допомогою бібліотеки libatm. Декілька програм можна пов'язати з цією бібліотекою для одночасної роботи і всі вони будуть напрямлені на один модуль atmsocket в ядрі. Для того, щоб здійснювалась взаємодія між libatm і atmsocket, ППІД АРП повинен перейти межі між ядром і простором користувача Digital UNIX.

У даному відтворенні не вводилось нових системних викликів для взаємодії з ядром, тому що це призвело б до проблем появи нових вільних кроків роботи операційної системи. У такому випадку ППІД АРП повинні бути повторно інтегровані в кожній новій версії системи. Натомість, ППІД використовує концепцію стандартизованого UNIX-сокета як інтерфейс до ядра. Однак ППІД АРП не реалізує сокетний інтерфейс АРП. Сокет тільки з'єднує виклики функцій з бібліотеки libatm в модуль atmsocket і навпаки. То ж позитивним є те, що функціональність наведеного ППІД можна легко розширювати.

Цей підхід аналогічний до Fore API [9] який також використовує інтерфейс сокетів BSD або потоковий інтерфейс System V, як канал зв'язку між API і драйвером пристрою. Більшість інтерфейсних функцій отримані безпосередньо з відповідних примітивів семантики опису форуму АРП [3,4]. Більш детально, ППІД АРП пропонує виклики функцій з програми (libatm) до модуля atmsocket, які називатимемо викликами на нижчий рівень "downcalls" і зворотного виклику функцій для індикації, наданої від модуля atmsocket до додатка, які теж назвемо викликами на вищий рівень "upcalls". Передбачені downcall-функції, можуть бути згруповані в певні категорії, де кожна категорія міститиме функції, що пов'язані з локальними завданнями, такими як створення або видалення локального

ідентифікатора зв'язку з кінцевою точкою з'єднання (КТЗ) (наприклад, такою як *ATM_associate_endpoint()*).

Іншими категоріями є, відповідно, встановлення, звільнення та переривання вихідних або вхідних з'єднань (наприклад, таких як *ATM_connect_outgoing_call()*). Крім того, дані надсилаються функцією *ATM_send_data()*, а інші функції взаємодіють з індикаторами та підтвердженнями, які обробляються інтерфейсом (наприклад, *ATM_process()*, як було вже описано). Аналогічним чином, upcalls-виклики, які додаток може зареєструвати в ПІПД, можуть також бути згруповані в класи. Є спеціальні функції для повідомлення додатка про отримання виклику або надходження даних. Крім того, є зворотні або callback-виклики, які підтверджують завершення або відмову від операції як додавання частки в груповий зв'язок (яка не підтримується до версії Digital UNIX 4,0e). Також повідомляється, коли з'єднання готове для обробки даних, що будуть передаватися. І нарешті, також є функція, яка вказує на завершення виклику.

Реалізація ПІПД базується на одній нитці виконання. Щоб уникнути блокування додатка, коли, наприклад, він очікує вхідний виклик, всі downcall-виклики ПІПД були реалізовані як неблокуючі функції. Для того, щоб дозволити можливість ПІПД виконувати upcall-виклики до додатка, має бути виділена нитка виконання. Це робиться шляхом виклику downcall-функції ATM_process() з відповідним значенням для кінцевої точки з'єднання (КТЗ). В залежності від показників очікування API далі викликає відповідну upcall-функцію і, таким чином, виступає в якості диспетчера. Додатку, який спонукає асинхронний виклик upcall-функцій, потрібно створити ще один потік, який блокується до тих пір, поки сокет, який пов'язаний з КТЗ, згенерує подію.

Спроектований ПІПД АРП був використаний для реалізації різних рівнів протоколів над АРП [8]. Спеціальний механізм моніторингу QoS був інтегрований в ПІПД [10], а контроль якості обслуговування QoS підтримувався новими послугами спеціального підрівня конвергенції ПСПК (або *Service Specific Convergence Sublayer*) протоколом для шару сервісу рівнів адаптації асинхронного режиму передачі загального використання. Такий QoS-моніторинг ПСПК об'єднує додаткову інформацію управління, таку як часові мітки і порядкові номери, в регулярний потік даних. За допомогою цієї інформації та спеціального підходу вирівнювання в протоколі трейлера, QoS-моніторинг ПСПК здатний підтримувати моніторинг шару рівнів адаптації асинхронного режиму передачі з кінця в кінець, а також моніторинг стрибків шару рівня асинхронної передачі. Крім того, такий механізі потужного підвищення контролю вже інтегрований в останню версію ПІПД [11] і може бути легко активований шляхом комунікації з типом ПСПК-сервісу нашого QoS-моніторингу ПСПК при встановленні з'єднання. Оцінка з кінця в кінець QoS-моніторингу інформації для ПСПК може бути виконана за допомогою спеціалізованого QoS-монітора [12], який в даний час інтегровано в модуль armsocket [11] в просторі ядра. Через це, детальні результати моніторингу на базі нового QoS-моніторингу ПСПК все ж не можуть бути отримані.

ПІПД для АРП Java в Digital UNIX

Насамперед було втілено ряд пропозицій для того, щоб надати доступ до функціональності АРП в Java [13]. Однією із таких пропозицій для розширення Java є пропозиція операцій з сокетами стилю BSD [14-16]. Іншою – є пропозиція розширення пакета java.net з метою імітувати функції Java-сокетів якомога ближче [17-20], для того щоб реалізувати функціональність, описану в роботі [4], яка пізніше була розширена в наступних роботах. Цей підхід був узятий на нашу Java-бібліотеку ПІПД, що реалізує основні функції специфікації для ядра і був успішно протестований в Java Development Kit (JDK) 1.1.7 і 1.2/2.0.

Для доступу до функцій "іншими" мовами в Java функціонує доступний власний Java-інтерфейс (BI Java). Інструмент Java-h генерує заголовки С-функцій із оголошень в Java-класі. Виклик методу функцією, оголошеного як власного в результатах Java, здійснюється у відповідній С-функції, що викликається. Однак, параметри передаються дещо інакше, ніж із С-функцій. Замість передачі параметрів безпосередньо, деякі вказівники на, наприклад, середовище Java, передаються на функцію, яка, в свою чергу, повинна знайти відповідні фактичні змінні. Це означає, що не можливо прямим способом викликати функції регулярної бібліотеки С з Java в UNIX, але замість цього повинен бути введений проміжний шар, який, як мінімум, визначає параметри і відслідковує синтаксичні відмінності. Рисунок 2 зліва демонструє архітектуру ПІПД Java, у тому числі проміжний шар та ПІПД мови С. Потрібно бути обережним і не слід плутати "власні послуги АРП " і "власні функції інтерфейсу" в цьому контексті.

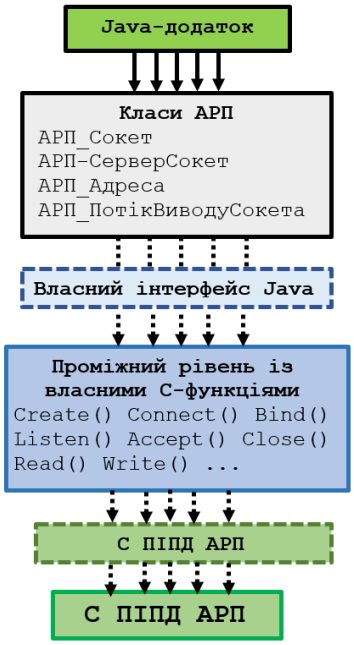

Рис. 2 – Архітектура ПІПД Java на Digital UNIX

AtmSocket створює клієнтський сокет, аналогічний сокету мови С, AtmServerSocket – це серверний сокет. Обоє вони використовують абстрактний клас AtmSocketImpl, який пропонує методи відповідно до "заводського" шаблону проектування, разом з класом PlainAtmSocketImpl. AtmSocketInputStream і AtmSocketOutputStream пропонує відповідно декілька функцій читання і запису, які відомі зі звичайних Java сокетів. Для вказівки адреси АРП використовується новий клас AtmAddress. Класи AtmBLLI і AtmBHLI представляють інформацію високого і низького рівня розширення АРП, відповідно. Параметри з'єднання визначаються за допомогою AtmConnAttr, але в даний час – ігноруються.

Оцінка продуктивності

Адаптери АРП, що використовувалися, пропонують лінійну швидкість ~155 Мбіт/с. Віднімаючи інформацію протоколу і управління, яка передається, шар АРП пропонує підтримку швидкості 135,63 Мбіт/с. Пропускна здатність, втрати блоків (або комірок) і отримані графіки системного навантаження залежать від багатьох факторів, а найбільше від розміру посланих блоків.

Використовуючи рівень потужності Digital UNIX в 8,98 кбайт, застосування IP через АРП на робочих станціях Digital UNIX дає продуктивність близько 60 Мбіт/с. ПІПД АРП, який ще не був оптимізований за швидкістю, досягає пропускної здатності від 85 до більш ніж 120 Мбіт/с при утворенні зв'язку між двома різними машинами. Низька продуктивність IP через АРП пояснюється його більш високими затратами на обробку протоколу. Якщо використовувати для двохстороннього з'єднання свіч АРП до однієї і тієї ж машини, то пропускна здатність через власний інтерфейс досягає ~40 Мбіт/с. У цьому випадку частина ядра АРП займає ~90% циклів процесора. Java ПІПД на таких же машинах досягає пропускної здатності відсилання до ~106 Мбіт/с (але з дуже високим рівнем втрат приймача, які, ймовірно, обумовлені неоптимальністю буфера обробки на приймальній стороні), при використанні з однієї машини на іншу і до 85 Мбіт/с (при значно нижчому рівні втрат) при відправці до свіча і назад до тієї же машини. Ці цифри нижчі, ніж для ПІПД С через додаткові затрати, пов'язані з проміжним шаром, а високий рівень втрат, пов'язаний з рівнем швидкості надсилання до 90 Мбіт/с спричиняє значно менший рівень отримування (rate2). З використанням удосконаленої віртуальної машини Java-fast можна збільшити вихідну досягнуту пропускну здатність за допомогою збільшеного зображення пам'яті (44 Мб замість 11 Мб) і вищого рівня втрат (імовірно приріст операції надсилання зростає швидше, ніж здійснення отримання). Використання повільнішого PrintStream замість DataOutputStream стимулює зростання пропускної здатності від 5 до 8 Мбіт/с при втратах близько 0,01%. Отже пропускна здатність сильно залежить від розміру блоку і значення вихідного рівня.

Висновок

В роботі були представлені три різні підходи для реалізації власного інтерфейсу АРП. У той час як реалізація ПІПД АРП мови С повністю інтегрована в архітектуру цифрового модуля управління з'єднанням (Connection Management Module), ПІПД Java був реалізований на вершині ПІПД мови С і якому необхідний проміжний рівень. Із-за різних архітектур (і складнощів), а також через різницю характеристик продуктивності основних машин, операційних систем і драйверів досягнуті в роботі значення пропускних здатностнй також відрізняються. Отримані цифрові значення збільшують впевненість, що є можливість налаштування коду, так щоб використовуючи сучасні машини, можна було б скористатися перевагами АРП за ради збільшення пропускної здатності при умові забезпечення відповідної підтримки драйверів для цього функціоналу.

1. M. Laubach: Classical IP and ARP over ATM, Request for Comments 1577, Internet Engineering Task Force (IETF), January 1994.
2. The ATM Forum: LAN Emulation over ATM, Version 2 - LUNI Specification; af-lane-0084.000, July 1997.
3. The ATM Forum: Native ATM Services: Semantic Description Version 1.0; af-saa-0048.000, February, 1996.
4. The ATM Forum: API Semantics for Native ATM Services using UN1 4.0; af-saa-0108.000, December 1998.
5. W. Almesberger: Linux ATM API, Draft, version 0.4, EPFL, LRC, Switzerland, http://lrcwww.epfl.ch/linux-atm/, July 19, 1996.
6. S. Keshav, H. Saran: Semantics and Implementation of a Native-Mode ATM Protocol Stack; AT&T Bell Laboratories Technical Memorandum, 1994, http://www.cs.att.com/csrc/keshav/papers.html
7. R. Ahuja, ***S.*** Keshav, H. Saran: Design, Implementation and Performance of a Native Mode ATM Transport Layer; Proc. IEEE INFOCOM, March 1996.
8. Stefan Dresler, Markus Hofmann, Claudia Schmidt, Hajo R. Wiltfang: A Native ATM API Suited for Multimedia Communication; Fourth International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS P7), Darmstadt, Germany, September 1997.
9. Edoardo Biagioni, Eric Cooper, Robert Sansom (Fore Systems Inc.): Designing a Practical ATM LAN; IEEE Network, March 1993 http://www.cs.att.com/csrc/keshav/papers.html.
10. H. R. Wiltfang, C. Schmidt: QoS Monitoring for ATM-based Networks; in Proceedings of the International Conference on Management of Multimedia Networks and Services; Montreal, Canada; July 8- 10, 1997.
11. S. Dresler, R. Lisak, H. Ritter, H.R. Wiltfang. Native ATM Interfaces in C and Java: Implementation and Experiences. Lecture notes in computer science 1309. – 1997. – Springer. – p. 352-471.
12. C. Schmidt, R. Bless: QoS Monitoring in High Peflorniance Environments, in Proceedings of the Fourth IFIP International Workshop on Quality of Service; Paris, France; March 6-8, 1996.
13. The ATM Forum/T. Jepsen, J. Shaffer: Java ATM API Description-Proposed Outline, Revision 1; contribution atm97-1044r1; February 1997.
14. The ATM Forum/T. Jepsen, S.A. Wright: A Java Syntax Instantiation of the Native ATM Services; contribution atm97-0772, September 1997.
15. The ATM Forum/T. Jepsen, S.A. Wright: A Java Syntax Mapping to the Native ATM Services; contribution atm97-0773, September 1997.
16. The ATM Forum/T. Jepsen, S.A. Wright: A Java Syntax Implementation of the Native ATM *Services;* contribution atm97-0774, September 1997.
17. The ATM Forum/J. Harford: Java ATMAPI from 50,000 Feet; contribution atm97-1110, December 1997.
18. The ATM Forum/J. Sung: Extension of the java.net package to support Native ATM Services, Revision 1; contribution atm98-0332r1, July 1998.
19. Microsoft: *Windows Sockets* 2 *API, Revision 2.2.0;* May 10, 1996.
20. The ATM Forum/Tom Jepsen, John Shaffer: *Java ATM API Working Document-Baseline Text;* contribution btd-saa-api-Java-00.03, April 1999.