УДК 681.3.06

О.М. САКАДА, О.І. МАРЧЕНКО

Національний технічний університет України "КПІ", Україна

СТРУКТУРА N-ВЕРСІЙНОГО ПЛАНУВАЛЬНИКА

В статті аналізується архітектура програмного забезпечення, яке розробляється для вбудованих систем з використанням концепції N-версійного програмування. При цьому вважається, що операційна система, однією з найбільш важливих компонент якої є планувальник, входить до складу програмного забезпечення. В статті особлива увага приділена структурі планувальника, який виконує передачу управління між програмними модулями, що розроблені згідно концепції N-версійного програмування. Крім того, сам планувальник також розроблений згідно даної концепції. В статті описується метод передачі управління між екземплярами планувальника.

N-версійне програмування, структура, планувальник, управління, задача

Питання, пов'язані з надійним функціонуванням апаратно-програмного забезпечення, мають пріоритетне значення при розробці стійких до відмов вбудованих систем. Розробка програмного забезпечення проходить різні етапи, починаючи від проектування і закінчуючи тестуванням та налагодженням коду. Але людський фактор має вплив на кожний етап цього складного процесу. Тому навіть після завершення тестування не можна виключати можливості знаходження помилок в програмному коді.

Для вирішення даної проблеми, була запропонована концепція N-версійного програмування, яка передбачає реалізацію декількох версій одного і того ж алгоритму незалежними групами програмістів [1 – 4]. Також передбачається реалізація додаткового модуля, який порівнює вихідні дані попередніх модулів між собою під час виконання. Але не до кінця вирішеною задачею є той факт, що вибираються не всі, а лише певні програмні модулі, для яких реалізується більш ніж одна версія. Можливі помилки в коді решти модулів залишаються в такому випадку непоміченими, навіть під час виконання [5, 6].

Автори статті пропонують розглянути можливість поширення застосування концепції N-версійного програмування не вибірково, а на все програмне забезпечення. В даній статті розглядається можливість такого поширення на планувальник.

Модель помилок

Перш за все, необхідно розрізнити два терміни – помилка в програмному коді, та наслідки наявності та прояву помилки у вигляді неправильної, неочікуваної або помилкової поведінки програмного коду. Це принципово різні речі. В більшості випадків необхідно та достатньо виявити саме наслідки помилки. Це робиться під час виконання програми. Знаходження самої помилки є другорядною задачею. Така задача може вирішуватися в будь-який час, не обов'язково під час виконання.

Тому, визначимо наслідки прояву помилки, які повинні виявлятися програмним забезпеченням. Це неправильно пораховані або пошкоджені дані, адреси, та/або дані, що використовуються в операторах управління (напр. умовних операторах, операторах циклу, т.п.). Це може бути наслідком прояву помилки в тій версії програмного забезпечення, якій пошкоджені дані належать, або наслідком помилки яка знаходиться в будь-якій іншій версії програмного забезпечення (крім пошкодженої).

Помилкова поведінка в першому випадку легко знаходиться на етапі виконання програмного забезпечення, розробленого із застосуванням концепції N-версійного програмування. При $N > 2$, також достатньо легко вказати на місцезнаходження помилки

(з точністю до версії програмного забезпечення).

Вирішення проблеми в другому випадку лише програмними засобами є не типовим. Також, в більшості випадків навіть при $N >> 2$ неможливо встановити місцезнаходження помилки (навіть з точністю до версії програмного забезпечення). Тому, є доцільним використати додаткові архітектурні рішення та програмно-апаратні засоби.

По-перше, кожна реалізована версія програмного забезпечення має бути спроектована таким чином, щоб не виконувати доступ на запис до даних, що знаходяться в області пам'яті, яка належить іншій версії.

По-друге, необхідність наявності в системі (в даному контексті під системою слід розуміти програмно-апаратне забезпечення) апаратного пристрою захисту пам'яті. Так, наприклад, кожний такий пристрій може захищати відповідну версію програмного забезпечення: коли управління передається певній версії, то відповідний пристрій деактивується; коли певна версія реалізації втрачає управління – відповідний пристрій активується, захищаючи таким чином дані від можливого пошкодження.

Структура програмного забезпечення

Наведемо перелік умов, які накладемо на програмне забезпечення, для якого аналізується можливість та розробляється спосіб застосування N-версійного програмування:

1. Програмне забезпечення розробляється для вбудованих систем. Для таких систем характерні наведені нижче властивості (перелічені лише ті властивості, що мають відношення до структури планувальника):

– замкнена циклічна обробка даних, яка передбачає доступ до апаратної частини для отримання вхідних даних, обробку отриманих даних, а також доступ до апаратної частини для запису оброблених даних;

– незмінна форма направленого графа програмного забезпечення (вершинами якого є програмні модулі, а ланками – напрямок передачі даних від одного програмного модуля до іншого); як наслідок, кількість модулів теж незмінна.

2. Програмне забезпечення містить в своєму складі операційну систему. Наявність операційної системи характерне для більшості вбудованих відмовостійких систем.

В даній статті розглядається принцип побудови однієї з найбільш важливих компонент операційної системи – планувальника. Вважається, що всі програмні модулі (крім планувальника) розроблені згідно концепції N-версійного програмування. Тому вимоги для планувальника визначені нижче.

1. Планувальник повинен керувати передачею управління між програмними модулями, які розроблені згідно концепції N-версійного програмування.

2. Сам планувальник повинен бути розроблений згідно даній концепції.

Типова структура вбудованого програмного забезпечення показана на рис. 1. Модуль А зчитує вхідні дані з певного апаратного пристрою, передає їх на обробку в модуль В. Модуль В оброблює дані, отримані з модуля А, та передає їх наступному модулю. Модуль М, який є останнім в ланці програмних модулів, виконує завершальну обробку даних та записує їх в певний апаратний пристрій.

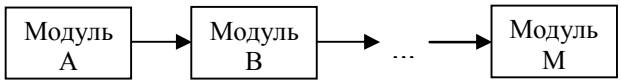

Рис. 1. Типова структура вбудованого програмного забезпечення

Після того, як модуль М виконав свою роботу, управління передається знову модулю А. Цей процес відбувається впродовж всього часу роботи вбудованої системи.

Якщо застосувати концепцію N-версійного програмування до наведеної на рис. 1 системи, то відповідна структура буде мати вигляд, що показаний на рис. 2.

Там показано, що існує N версій реалізації алгоритму для кожного програмного модуля. Передача управління між модулями відбувається за схемою, схожою на ту, що відповідає системі, яка показана на рис. 1. Але існує декілька відмінностей, на які варто звернути увагу.

По-перше, передача управління до модулів, які знаходяться на наступному рівні, здійснюється лише після того, як виконали свою роботу всі модулі що знаходяться на поточному рівні. Модулі, що реалізу-

ють один і той же алгоритм, належать до одного рівня. Так, наприклад, управління буде передано одному з модулів B1, B2, ... BN лише після того, як виконав свою роботу кожний з модулів A1, A2, ... AN.

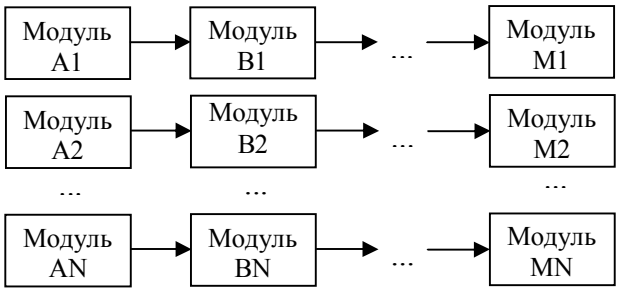

Рис. 2. Типова структура вбудованого програмного забезпечення розробленого згідно концепції N-версійного програмування

По-друге, концепція N-версійного програмування передбачає наявність в системі додаткових, порівнюючих, програмних модулів. Так, наприклад, на місці модулів B1, B2, ... BN повинні бути різні реалізації модуля, який порівнює вихідні дані модулів A1, A2, ... AN між собою. Тому при застосуванні концепції N-версійного програмування кількість модулів збільшується не лише за рахунок реалізації N версій кожного модуля, а також і за рахунок наявності додаткових порівнюючих, для кожного з яких також передбачається N версій реалізації. Зі структурної точки зору, це не має значення, бо, скажімо, обчислювальні та порівнюючі модулі реалізують різні за принципом алгоритми, а тому належать до різних рівнів. А значить, не змінюють принцип передачі управління.

Структура планувальника

Структура планувальника повинна задовольняти раніше сформульованим вимогам:

1. Планувальник повинен бути розроблений із застосовуванням концепції N-версійного програмування. Це означає, що повинно бути реалізовано *N* версій (назвемо екземплярів) планувальника.

2. При взаємодії різних екземплярів планувальника між собою (тобто при передачі управління від одного екземпляру до іншого) жодний з екземплярів не повинен виконувати доступ на запис до даних, які належать іншому екземпляру.

Передачею управління між якими модулями, а точніше екземплярами модулів, повинен керувати кожний з екземплярів планувальника? Відповідь однозначна: *j*-й екземпляр планувальника повинен керувати передачею управління між *j*-ю версією реалізації кожного з модулів, де *j* – номер версії реалізації відповідного модуля та змінюється в межах $1 \le j \le N$.

Такий підхід забезпечує стійке функціонування системи, навіть при прояві помилки, внаслідок якої певний екземпляр планувальника не зможе правильно керувати (або взагалі керувати) передачею управління між програмними модулями. Крім того, такий підхід є прямим поширенням концепції N-версійного програмування на все програмне забезпечення: реалізується *N* версій одного і того ж програмного забезпечення, яке включає в себе крім обчислювальних та порівняльних модулів ще й планувальник. Структурна схема такого програмного забезпечення показана на рис. 3.

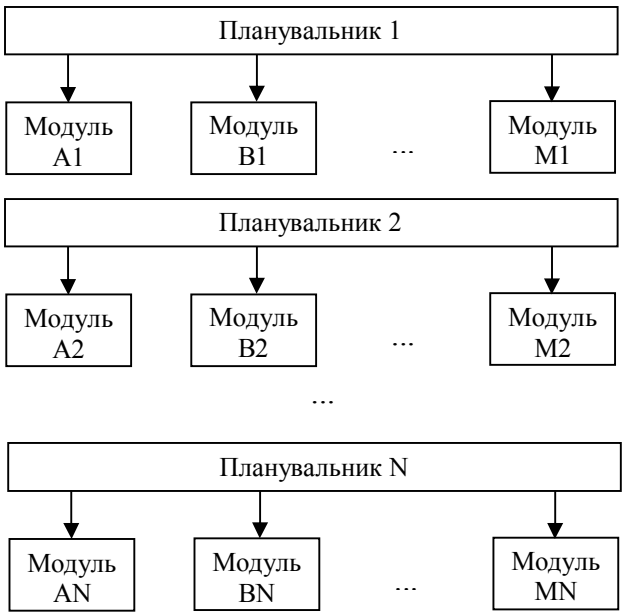

Рис. 3. Структура програмного забезпечення, до складу якого входить планувальник і яке розроблене згідно концепції N-версійного програмування

Щоб задовольнити вимогу, сформульовану в пункті 2 даного розділу, необхідно зрозуміти спосіб, завдяки якому планувальник отримує управління, і в свою чергу, передає управління певній задачі.

Як правило, в склад мікропроцесора, на якому виконується програмне забезпечення, входить один або декілька таймерів. Один з таких таймерів вико-

ристовується для генерації апаратних переривань, які є тим механізмом, завдяки якому управління переходить від поточної задачі до планувальника.

Після генерації таймером переривання, управління передається відповідному програмному обробнику переривань. Даний обробник переривань таймера є невід'ємною частиною планувальника.

Як правило, обробник переривань виконує такі операції:

– збереження в оперативній пам'яті контексту поточної (перерваної) задачі;

– знаходження наступної задачі, якій необхідно передати управління;

– встановлення таймера на відповідний до пріоритету задачі квант часу;

– завантаження в регістри процесора раніше збереженого контексту даної задачі (цією дією починається виконання даної задачі в межах виділеного кванту часу).

Якщо принцип передачі управління між задачами кожним окремим екземпляром планувальника достатньо зрозумілий, то принцип передачі управління між екземплярами планувальника потребує певного аналізу.

Оскільки є необхідність реалізувати *N* версій планувальника, то необхідно відповідно реалізувати таку ж кількість і обробників переривань таймерів. Якщо використовувати тільки один таймер, то необхідно буде кожний раз перезаписувати адреси обробника переривань у таблиці обробників переривань. А це означає, що буде існувати область пам'яті, до якої повинні мати доступ на запис всі екземпляри планувальника. Це, в свою чергу, суперечить одній з умов, яка була сформульована в розділі «Модель помилок». Більш того, передача управління таким чином між екземплярами планувальника передбачає, що кожний екземпляр повинен володіти інформацією про наступного екземпляра, якому повинно бути передане управління, для того щоб записати його адресу у таблицю обробників переривань. Така залежність має негативний вплив на відмовостійкість системи.

Таким чином, необхідна наявність в процесорі (або в апаратному забезпеченні у випадку використання зовнішніх таймерів) N таймерів. Кожний з цих таймерів буде використовуватись тільки відповідним екземпляром планувальника. Обробник переривань у кожного таймера теж буде свій і адреса кожного обробника буде занесена у таблицю лише один раз при конфігурації системи.

Яким саме чином буде передаватись управління між екземплярами планувальника? Перш за все, це залежить від співвідношення запланованої кількості процесорів та кількості розроблених версій програмних модулів. Існує два варіанти.

Якщо кількість процесорів дорівнює кількості версій програмних модулів, то в цьому випадку кожний екземпляр планувальника разом із відповідними версіями програмних модулів може виконуватись на окремому процесорі. При цьому немає потреби контролювати передачу управління між різними екземплярами планувальника. Але кожний екземпляр планувальника повинен передати управління модулю, що знаходиться на наступному рівні, тільки після того, як завершили своє виконання на різних процесорах всі екземпляри модуля, що знаходиться на поточному рівні. Для цього в планувальнику повинні бути реалізовані елементи синхронізації.

Якщо кількість процесорів менша кількості версій програмних модулів, то в цьому випадку, принаймі на одному з процесорів, буде виконуватись більш ніж одна версія планувальника з відповідними екземплярами модулів. Умови передачі управління від одного модуля до іншого такі ж самі, як і в попередньому пункті. Управління між екземплярами планувальників в даному випадку буде передаватись за допомогою методу мультиплексації переривань таймерів. Суть даного методу полягає в наступному.

Кожний екземпляр планувальника оброблює два суміжні переривання відповідного таймера. Непарні переривання планувальник використовує для знаходження наступної задачі, якій необхідно передати управління, встановлення таймера на відповідний до пріоритету задачі квант часу, а також завантаження в регістри процесора раніше збереженого контексту даної задачі. Парні переривання планувальник використовує для збереження в оперативній пам'яті (що належить даному екземпляру планувальника) контексту поточної (перерваної) задачі та передачі управління задачі idle. Дана задача виконує ключову

роль в передачі управління між екземплярами планувальника – між парним та непарним перериваннями різних таймерів її завданням є не виконувати ніяких дій. Саме завдяки наявності задачі idle зникає потреба безпосередньої передачі управління між екземплярами планувальника.

Нехай $T_i^{парне}$ – це час виникнення парного переривання таймера і, а $T_i^{непарне}$ – час виникнення непарного переривання таймера і. Тоді таймери повинні бути сконфігуровані таким чином, щоб час виникнення відповідних переривань задовольняв співвідношенню $T_i^{парне} > T_i^{непарне} > T_{i-1}^{парне}$, для $1 \le i \le Np$, де Np – кількість екземплярів планувальників (а відповідно і задіяних таймерів) що виконуються на одному процесорі p.

На рис. 4 в якості прикладу наведена часова діаграма даного методу для випадку, коли кількість екземплярів планувальника, що виконуються на одному процесорі, дорівнює 2.

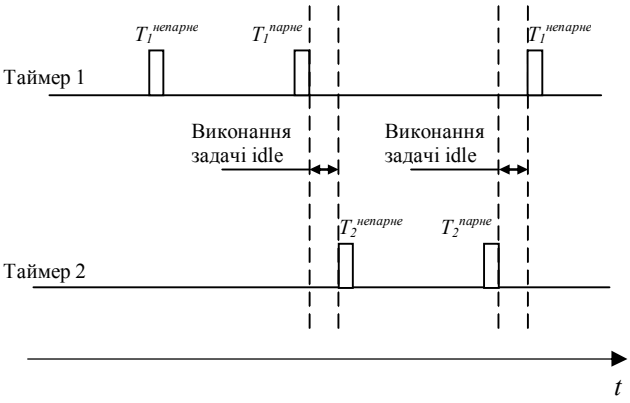

Рис. 4. Часова діаграма методу мультиплексації переривань таймерів для випадку, коли кількість таймерів дорівнює 2

Розглянемо дві події, що показані на рис. 4 – $T_1^{парне}$ та $T_2^{непарне}$. При виникненні переривання $T_1^{парне}$, екземпляр 1 планувальника повинен:

1) завершити виконання поточної задачі (версії 1);

2) активувати відповідний пристрій захисту пам'яті для збереження власних даних від несанкціонованого доступу під час виконання іншого екземпляра планувальника та відповідних програмних модулів;

3) передати управління задачі idle, чекаючи появу переривання $T_1^{непарне}$.

Все що відбуватиметься в системі до настання події $T_1^{непарне}$, буде відбуватися поза контролем екземпляру 1. В цей час виникне переривання $T_2^{непарне}$, для обробки якого управління буде передано екземпляру 2. Він виконає такі дії:

1) деактивує відповідний пристрій захисту пам'яті;

2) знайде задачу (версії 2) якій необхідно передати управління;

3) завантажить в регістри процесора раніше збережений контекст даної версії задачі.

Таким чином, в статті була визначена модель помилок, було запропоновано та розглянуто одну з можливих структур планувальника, який придатний до виявлення відповідних помилкових ситуацій та розроблений із застосуванням концепції N-версійного програмування. Отриманий результат підтверджує принципову можливість реалізації такого підходу та може бути використаний для розробки відмовостійких вбудованих систем.

Література

1. Laura L. Pullum, Software Fault Tolerance Techniques and Implementation. – Norwood, Artech House, 2001.

2. Chen Y. Operating Systems for Safety-Critical Applications, Electron // Journal of the South African IEE. – January 2000. – P. 47-48.

3. Salles F., Arlat J., Fabre J-C. Can we rely on COTS microkernels for building fault-tolerant systems? – The 33rd Meeting of IFIP 10.4 WG, Cape Town. – January 1998. – P. 13-20.

4. Chen Y. Modeling Software Operational Reliability under Partition Testing // IEEE 28th Annual International Symposium on Fault-Tolerant Computing (FTCS-28), Munich. – June 1998. – P. 314-323

5. Симоненко В.П. Организация вычислительных процессов в ЭВМ, комплексах, сетях и системах. – К.; ВЕК+, 1997. – 307 с.

6. Петренко А.К. Методы отладки и мониторинга параллельных программ // Программирование. – 1994. – №3. – С. 39-63.

Надійшла до редакції 28.02.2006

Рецензент: канд. техн. наук, доцент О.А. Щербина, Київський національний університет будівництва та архітектури.