УДК 625.039

Т.І. ЄФІМОВА, Б.Г. МУДЛА, О.М. ШАЛЕЙКО

ВІДМОВОСТІЙКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ГАРАНТОЗДАТНИХ КОМП'ЮТЕРНИХ СИСТЕМ

Abstract: *In the paper the problems of creation of the fault-tolerant software for dependable computing systems are considered. The analysis of a modern status of a problem is carried out. The various methods of creation of the fault-tolerant software are described. The ratings of the approaches of the N -version software and model of recovery blocks are made. The example of the concrete project of the N -version software is given.*
Key words: *dependability, software, error, fault, failure, fault tolerance, specification, N -version software.*

Анотація: *У статті розглянуті проблеми створення стійкого до відмов програмного забезпечення для гарантоздатних бортових автономних комплексів довготривалого функціонування. Проведено аналіз сучасного стану проблеми. Описані різні методи створення відмовостійкого програмного забезпечення. Зроблені оцінки підходів N -версійного програмного забезпечення та моделі блоків відновлення. Приведений приклад конкретного проекту N -версійного програмного забезпечення.*
Ключові слова: *гарантоздатність, програмне забезпечення, помилка, несправність, відмова, відмовостійкість, специфікація, N -версійне програмне забезпечення.*

Аннотация: *В статье рассмотрены проблемы создания отказоустойчивого программного обеспечения для гарантоспособных бортовых автономных комплексов длительного функционирования. Проведен анализ современного состояния проблемы. Описаны различные методы создания отказоустойчивого программного обеспечения. Сделаны оценки подходов N -версионного программного обеспечения и модели блоков восстановления. Приведен пример конкретного проекта N -версионного программного обеспечения.*
Ключевые слова: *гарантоспособность, программное обеспечение, ошибка, неисправность, отказ, отказоустойчивость, N -версионное программное обеспечение.*

1. Вступ

Міжнародна систематизація базових понять і таксономія гарантоздатних (надійних та безпечних) обчислень та їх модифікована таксономічна схема з урахуванням факторів еволюції комп'ютерних систем (КС) досліджуються в роботах [1, 2]. В них та у роботі [3] гарантоздатність визначається як надскладна властивість системи гарантовано надавати задані специфікацією послуги (виконувати функції), яким можна виправдано довіряти; гарантоздатною є система, якій притаманна така властивість. Програмне забезпечення (ПЗ) сучасних гарантоздатних КС базується на готовності, безвідмовності, функціональній безпеці, цілісності та обслуговуванні. В той же час воно повинно відповідати вимогам безпеки. Як відомо із [4, 5], безпека є комбінацією конфіденційності (попередження неправомірного розкриття інформації), цілісності (попередження неправомірного правлення або стирання інформації) та готовності (попередження неправомірної відмови поставки інформації). В роботі [6] доведено, що показники гарантоздатності та безпеки слід розглядати як різні аспекти загальної проблеми. Це означає, що до них можна застосовувати загальні рішення. Отже, підходи, що базуються на принципах відмовостійкості, можуть забезпечити високий рівень як безвідмовності, так і безпеки.

2. Способи реалізації відмовостійкості програмного забезпечення

Відмовостійкість (стійкість до відмов) є фундаментальним методом для досягнення гарантоздатності обчислень. Способами забезпечення відмовостійкості є:

– виявлення помилки;

– обробка несправності;

– оцінка пошкодження;

– відновлення після помилки.

Всі ці способи реалізуються за допомогою захисної надлишковості (надмірності) алгоритмічних, функціональних, часових та інших елементів архітектури системи з урахуванням обмежень, обумовлених несправностями та змінами вимог і умов використання.

2.1. Виявлення помилки

Виявлення помилки проводиться за допомогою перевірки правильності результату рішення завдання. В ідеалі правильність роботи системи повинна базуватись на перевірці, чи відповідає ця робота специфікації системи. Така перевірка повинна виконуватись незалежними від системи механізмами (блоками, модулями). Специфікація повинна бути виражена в термінах інформації, зовнішньої до системи. Однією із головних особливостей відмовостійкості ПЗ є можливість перевірки правильності результату до того, як результат вийде за межі системи (блока, модуля).

В залежності від обмежень вартості та продуктивності системи перевірки можуть виконуватись порівнянням результатів:

– повторного використання системи;

– використання кількох копій однієї і тієї ж системи;

– використання кількох різних версій системи.

Іноді застосовують перевірку зворотним ходом: результат, отриманий системою, обробляється у зворотному порядку, щоб отримати відповідні входи і порівняти їх з фактичними. Ще один спосіб – перевірка результату на прийнятність. Використовують також відсутність відповіді компоненти на повідомлення в межах визначеного часу, порівняння контрольних сум інформаційних блоків тощо.

2.2. Обробка несправності

Обробка несправності полягає у визначенні місцеположення несправної компоненти ПЗ та заміні її на справний. Для цієї заміни система повинна мати у своєму розпорядженні надлишкові компоненти ПЗ та резерви часу. Надлишковість у системі може бути маскуюча або динамічна. Маскуюча надлишковість – статична, наприклад, потрійне модульне резервування (TMR). Динамічна надлишковість – це внутрішня надмірність компоненти, що використовується для визначення помилкової інформації. Вона повинна бути доповнена зовнішньою надлишковістю, яка забезпечить відновлення компоненти після помилки. У відновлюванні застосовують стратегії заміщення та реконфігурації. При заміщенні замість несправної компоненти включається резервна компонента, яка не була активною. При реконфігурації функції несправної компоненти перекладаються на ті компоненти системи, що працюють успішно. При цьому загальна продуктивність системи дещо зменшується, що слід ретельно враховувати при проектуванні критичних систем. На відміну від апаратного заміщення в ПЗ резервна компонента заміняє основну лише на деякий час (для певної комбінації вхідних даних), після чого виконується спроба повернути до роботи основну компоненту.

Для забезпечення гарантоздатності система або її компоненти повинні мати механізми обмеження помилки, що дозволять заблокувати процес розповсюдження пошкодженої інформації із системи (компоненти). Після цього слід визначити ступінь пошкодження та можливих наслідків для оточуючого середовища і включити необхідні механізми проти негативного розвитку подій та аварії.

2.3. Оцінка пошкодження

Оцінка пошкодження, викликаного несправністю, визначає, які процеси і починаючи з якого моменту слід вважати неправильними, щоб повторно їх перезапустити. Така оцінка виконується за допомогою методу структурування системи, що забезпечує представлення роботи системи у вигляді елементарних дій [7]. Значно простіше вважати, починаючи з вірної контрольної точки, яка була зафіксована в пам'яті перед ушкодженням процесів визначеної області, щоб повторно перезапустити їх або, враховуючи наявний резерв реального часу, замінити їх незіпсованими дублюючими дану функцію.

2.4. Відновлення після помилки

Процес відновлення після помилки застосовує два основні методи: ретроспективне відновлення після помилки та відновлення без повернення до попереднього стану.

Ретроспективне відновлення після помилки базується на фіксації в пам'яті контрольних точок. Цей метод припускає незнання точного місцеположення несправності та наслідків її дії в системі. Блок відновлення включає деяку кількість резервних альтернативних алгоритмів (що називаються «замінами»), на яких перезапускаються потрібні процеси з даними, взятими з пам'яті контрольних точок. Отже, форма методу дуже проста, оскільки таке відновлення не потребує діагностування та знаходження місцеположення несправності. Але при цьому втрачається час вже виконаної роботи.

Відновлення без повернення до попереднього стану припускає точну ідентифікацію несправності і забезпечення механізму для її усунення. При цьому заново виконуються лише ті процеси, що були виконані неправильно, тобто цей метод більш ефективний, ніж попередній. Однак він прийнятний лише для простих несправностей та механізмів їх усунення.

3. N -версійне програмне забезпечення

Відомо, що в наш час найбільшу загрозу для сучасних обчислювальних систем представляють несправності проектування. Різке зростання складності програмного забезпечення призвело до того, що стало практично неможливим виявлення та видалення всіх несправностей проектування ПЗ до вводу системи в фазу використання. Наприклад, відносно недавно була виявлена несправність проектування в мікропроцесорі Pentium, з яким уже були виготовлені та розповсюджені тисячі комп'ютерів. Ця несправність коштувала фірмі-виробнику $475 мільйонів. Таких прикладів можна привести безліч. Фундаментальним рішенням проблеми несправностей проектування в ПЗ є принцип різноманітності проектів. Різноманітність проектів виникла як аналог дублювання апаратних засобів. Але в апаратурі застосовується просте копіювання каналів обчислення, бо воно є ефективним засобом маскування випадкових фізичних несправностей

апаратних засобів. Проте копіювання блока ПЗ буде копіювати і не виявлені в ньому несправності проектування. Тому кожен блок ПЗ, що дублює виконання однієї і тієї ж функції, повинен бути розроблений незалежно від інших блоків. У цьому і полягає концепція різноманітності проектування програмного забезпечення або багатоверсійних обчислень.

Блок ПЗ, не забезпечений засобами відмовостійкості, називається симплексним блоком. Він навіть після тестування може мати невиявлені бездіяльні несправності, які при визначених обставинах (наприклад, при деякому випадковому наборі вхідних даних) активізуються і приводять до вироблення блоком неправильних вихідних результатів. У цьому випадку ми матимемо несправний блок ПЗ. Щоб зробити такий блок відмовостійким, до нього добавляють один або кілька симплексних блоків, які будуть маскувати результат несправного блока. Для того, щоб забезпечити відмовостійкість набору з N блоків, де $N \geq 2$, треба створити для нього середовище відмовостійкого виконання. Це середовище повинно забезпечувати порівняння результатів роботи блоків у спеціально вибраних точках перехресного контролю. Існує дві основні моделі побудови ПЗ із симплексних блоків: модель N-версійного програмного забезпечення (NVS) і модель блоків відновлення (RB) [8]. Ці моделі показані на рис. 1 та 2.

Рис. 1. Модель N-версійного програмного забезпечення (NVS) при $n = 3$

Обидві моделі застосовують різноманітність симплексних блоків. Ці блоки в NVS називаються версіями, а в моделі RB – альтернативами і приймально-здавальним тестом. Різниця моделей полягає в тому, як формується рішення відносно правильних виходів відмовостійкого блока, що складається з $N \geq 2$ симплексних блоків. Модель NVS застосовує алгоритм групового рішення, який забезпечує середовище виконання і знаходить консенсус двох або більшої кількості виходів серед N версій. Цей консенсус і буде правильним виходом відмовостійкого блока. Виходи розмножуються на N каналів для забезпечення роботи наступного відмовостійкого блока ПЗ.

Оскільки алгоритми, що використовують окремі версії, можуть бути різними, то і результати на виходах цих версій можуть лежати в деякому діапазоні значень. Тому необхідно мати алгоритм вибору рішення, щоб вибрати правильний результат з набору подібних, але не ідентичних результатів. Якщо декілька подібних результатів помилкові, вони називаються подібними

помилками. Якщо кількість подібних помилкових результатів перевершує кількість подібних правильних результатів у точці вибору, то алгоритм вибору рішення вибере помилковий результат. Наприклад, дві подібні помилки переважать один правильний результат у випадку трьох версій.

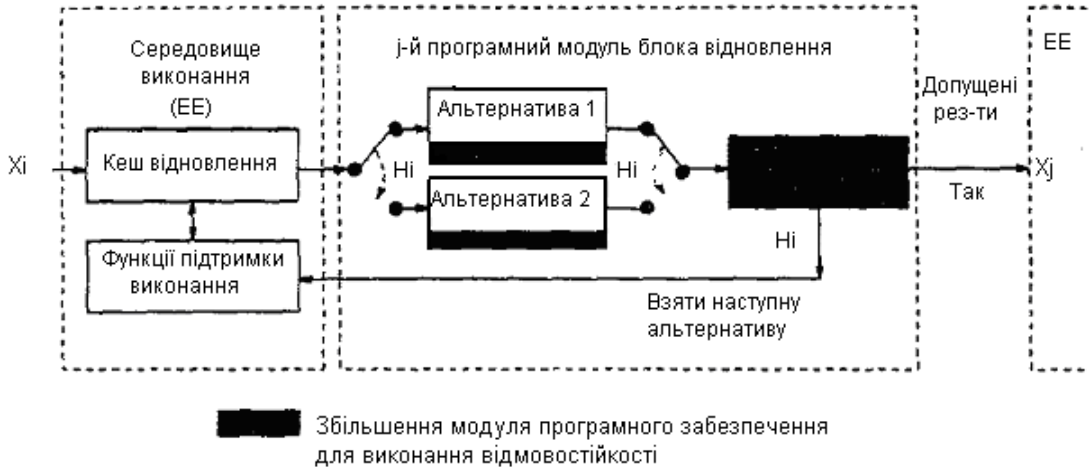

Рис. 2. Модель блока відновлення (RB)

Модель RB застосовує перевірку результату роботи індивідуальної альтернативи приймально-здавальним тестом. Цей тест має бути специфічним для даної програми і визначається замовником цієї програми. Якщо результат роботи альтернативи 1 пройшов перевірку приймально-здавальним тестом успішно, то його вихід вважається правильним і допускається на вхід наступного відмовостійкого блока ПЗ. В іншому випадку виконується заміна альтернативи 1 на альтернативу 2. Така модель у випадку виявлення несправності припускає повторне виконання блока на тій же апаратурі. Отже є більш затратною за часом та більш економною за апаратурою відносно моделі NVS.

Можливе застосування комбінації цих моделей. Наприклад, для прийняття рішення щодо правильності виходу окремої версії в моделі NVS можна скористатись приймально-здавальними тестами.

Як модель NVS, так і модель RB може бути реалізована для паралельного або послідовного виконання в залежності від наявності одночасно доступних апаратних каналів.

Для проектування індивідуальної версії або альтернативи потрібно виробити детальну специфікацію. Специфікація повинна повністю та однозначно сформулювати функціональні вимоги до програми і в той же час залишити максимальні можливості з різноманітності її реалізації. Специфікація повинна визначити:

– функцію, що повинна бути реалізована N-версією програмного модуля;

– точки перехресного контролю, в яких алгоритм вибору рішення буде визначати консенсус з іншими версіями;

– зміст і формат векторів перехресного контролю, які повинні генеруватися в кожній точці перехресного контролю;

– алгоритм вибору рішення, що повинен використовуватись у кожній точці перехресного контролю;

– відповідь на можливі результати рішення.

Для роботи ПЗ у режимі N -версійного виконання потрібен також механізм синхронізації. За допомогою цього механізму кожна версія сповіщає про готовність свого вектора та переходить у режим очікування сигналу від механізму вибору рішення. Механізм синхронізації також запобігає голосуванню результатів до того часу, поки не будуть готові результати (вектори) всіх версій.

Дії, які повинні бути виконані у відповідь на результати рішення в точках перехресного контролю, залежать від того, чи всі версії виробили відповідний результат (вектор) у межах заданого часу, та від того, чи співпадають ці результати. Такими діями можуть бути: продовження виконання версії, завершення виконання версії, продовження після заміни вектора дефектної версії на вектор, прийнятий у результаті рішення.

Алгоритм вибору рішення явно задає допустимий діапазон числових результатів, якщо він існує, та інші можливі різновиди результатів (наприклад, додаткові області текстового виводу). Найбільш перспективним підходом до розробки повної та правильної специфікації є використання формальних мов специфікації дуже високого рівня [9]. Наступним кроком являється незалежна розробка кількох специфікацій, що використовують різні формальні мови.

4. Дослідження відмовостійкості програмного забезпечення за допомогою ефективної різноманітності

В 1975–1978 роках в UCLA (Каліфорнійський університет в Лос-Анджелесі) проводились дослідження відносно вимог, які повинні виконуватися, щоб зробити N -версійне програмування можливим взагалі, незалежно від вартості. Також досліджувались методи для порівняння вартості та ефективності моделі NVS відносно одноверсійного програмування та підходу з блоком відновлення [9].

Дослідження з побудови відмовостійкого ПЗ на принципах N -версійності були розпочаті UCLA в 1975 році як частина робіт з безвідмовних обчислень. Накопичений з того часу досвід показано в табл. 1 [8]. Таблиця показує кількість розроблених версій, кількість програмістів у групі, що розробляла версію, яким чином планувалося забезпечення різноманітності та якими мовами програмування були реалізовані версії в роботах різних років.

Таблиця 1. Дослідження N -версійного програмування в UCLA

Час розробки	Проект та спонсор	Кількість версій	Розмір групи розробки	Потрібна різноманітність	Мови програмування
1975–1976	Текстовий редактор Група проектування програмного забезпечення	27	1	Персонал	PL/1
1977–1978	Рішення функції розподілення імовірностей Група проектування програмного забезпечення	16	2	Персонал та 3 алгоритми	PL/1
1979–1983	Розклад аеропорту Грант на дослідження NSF	18	1	Персонал та 3 специфікації	PL/1
1984–1986	Керування резервуванням датчиків Грант на дослідження NASA	5	2	Персонал	Pascal
1986–1988	Автоматичне приземлення літака Грант на дослідження від пілотних систем Sperry, Phoenix, AZ	6	2	Персонал та 6 мов	Pascal, Ada, Modula-2, C, Prolog, T

Найбільш важливою проблемою при проектуванні N -версійного ПЗ є:

– забезпечення того, щоб в окремих версіях не було взаємно пов'язаних несправностей проектування;

– мінімізація можливості того, що декілька незалежних версій у результаті несправності вироблять однакові помилкові виходи, які будуть сприйняті як консенсус правильних виходів.

Щоб запобігти такій ситуації, треба усунути всі можливі контакти між проектувальниками окремих версій програми.

Різноманітність проектів може бути досягнута випадково або відповідно до заданих вимог. Випадковою різноманітністю є така різноманітність, яка забезпечується незалежним персоналом проектувальників і викликана різницею в навчанні, способі мислення, досвіді роботи індивідуума і т.п. Потрібна різноманітність – така, яка досягається за рахунок різноманітності специфікацій, мов програмування, інструментальних засобів програмування, алгоритмів тощо. При виборі різноманітності в конкретній роботі слід враховувати вартість, терміни проектування та вимоги гарантоздатності.

5. Дослідження багатоверсійної системи програмного забезпечення на прикладі проекту програми керування автоматичним приземленням літака

Для дослідження відмовостійкості багатоверсійної системи ПЗ фірмами UCLA та Honeywell було розроблено науково-дослідний проект ПЗ, що забезпечує керування автоматичним приземленням літака [10]. В розробці було використано 6 версій програми, виконаних на 6-ти різних мовах програмування.

Метою дослідження були:

– розробка керівних принципів проектування для видалення причин пов'язаних несправностей у кількох незалежно розроблених версіях програми;

– пошук та детальне вивчення всіх потенційно пов'язаних несправностей у кількох незалежних версіях програми, що були розроблені на основі даної специфікації;

– розробка критеріїв якості, що дозволяють оцінити потенціал різноманітності через вивчення специфікації, з якої були створені версії;

– розробка методів вивчення набору версій з метою вивчення фактичної різноманітності в наборі та оцінка «випадкової» й «потрібної» різноманітності.

У роботі також проведено оцінку ефективності N -версійного ПЗ та його відносної безпеки в порівнянні з одиночною версією.

Фірма Honeywell, яка протягом більше 30 років була успішним розробником ПЗ керування польотом літака, забезпечила специфікацію програми керування автоматичним приземленням літака, тестові процедури, поставила модель літака та набори дослідних даних. Всі алгоритми та закони керування були визначені у відповідності до діаграм, які були завірені Федеральним управлінням авіації. Специфікація була відкоректована у відповідності до робочих параметрів визначеного літака і може бути використана в комп'ютері управління польотом реального літака. Цей документ визначає також «точки тестування», в яких повинні бути забезпечені виходи для додаткової перевірки на помилку. При написанні специфікації група координування керувалась

принципом поставки лише мінімальної інформації для програмістів, щоб запобігти небажаному впливу на розробку версій. Специфікація, яку отримали програмісти (разом з таблицями та рисунками), була написана англійською мовою на 64 сторінках. Її розробка велась групою координування, яку консультували спеціальні експерти, приблизно 10 тижнів. Розробка 6-ти версій ПЗ на 6-ти мовах програмування була виконана 12-ма програмістами протягом 12 тижнів. Отже, розробка специфікації та самого ПЗ забрала майже однакову кількість часу. Цей факт підкреслює виняткову важливість розробки специфікації.

Основним принципом різноманітності проектів було забезпечення максимально незалежних груп програмування. Це забезпечувало «випадкову» різноманітність. Але з метою введення «потрібної» різноманітності розглядались також різноманітність алгоритмів, мов програмування, середовищ розробки, інструментальних засобів. Різноманітність алгоритмів вирішено було не застосовувати з причини складності синхронізації версій та труднощів у гарантованому їх узгодженні. Була застосована різноманітність мов програмування, яка дозволяє запобігти помилкам компілятора. Це також дозволяє провести порівняльний аналіз мов високого рівня при застосуванні їх до одного і того ж алгоритму. Для реалізації версій були використані дві широко розповсюджені мови програмування: C та Paskal, дві об'єктно-орієнтовані мови: Ada та Modula-2, мова логічного програмування Prolog та мова функціонального програмування T, варіант Lips.

У процесі проектування дотримувались правила ізоляції груп програмістів з метою забезпечення відсутності взаємодії у процесі програмування. Для цього групам програмування для роботи були надані фізично розділені офіси. Зв'язок між групами був заборонений. Усі проблеми, пов'язані з роботою, вирішувала спеціальна група координування за допомогою електронної пошти. У спілкуванні з групою координування дотримувались принципу надання тільки необхідної інформації, щоб уникнути впливу на проектні рішення наданням надлишкової інформації.

Після розробки кожна версія програми пройшла стадію тестування. Тестування проводилось на базі технічних вимог. Воно складалося з трьох фаз: тестування модуля, комплексне тестування та приймально-здавальне тестування. У процесі тестування були виявлені та виправлені допущені несправності. Табл. 2 показує кількість виявлених у кожній версії несправностей та типи цих несправностей.

Таблиця 2. Класифікація несправностей у версіях за типами

Клас несправності	ADA	C	MODULA-2	PASCAL	PROLOG	T	Усього
Топографічна	0	1	0	0	9	0	10
Пропускання	1	3	0	0	8	5	17
Непотрібний код	1	0	0	2	0	2	5
Неправильний алгоритм	3	5	2	6	9	13	38
Неправильне тлумачення специфікації	1	3	1	4	0	1	10
Неоднозначність специфікації	0	1	0	0	0	0	1
Інші	0	0	1	0	0	0	1
Усього	6	13	4	12	26	21	82

Для перевірки роботи розроблених версій було згенеровано математичну модель літака та дій, що виконуються на фазі керування приземленням. У процесі тестування реєструвалась уся історія моделювання польоту. Потім порівнювались та аналізувались перетини польоту всіх версій.

У результаті цих дій виявлялись розбіжності та визначались несправності. Таким способом було проведено більш ніж 1000 експериментів моделювання польоту.

Основні висновки цього дослідження:

– керівні принципи генерації багатоверсійного ПЗ є досить повними та стійкими для використання в промисловому середовищі;

– первинна специфікація містила забагато інформації відносно проблем реалізації, що зумовило обмеження різноманітності;

– порядок обчислень, який розуміється в специфікації, має великий вплив на вибір алгоритмів програмістами навіть тоді, коли існують інші альтернативи; «точки тестування» теж обмежують різноманітність;

– застосування різних мов програмування сприяло ефективній ізоляції груп програмістів, оскільки використовувались різні інструментальні засоби підтримки; це сприяло різноманітності версій, що походили з однієї специфікації;

– відхилення від сформульованих правил проектування багатоверсійного ПЗ може привести до ідентичних і тому особливо небезпечних несправностей; строга верифікація цих правил проектування повинна бути частиною приймально-здавального тесту;

– подібні та співпадаючі в часі несправності (з причини ідентичності несправностей у двох версіях) були рідкісними; в 82 несправностях, що були видалені з 6 версій перед приймально-здавальним тестуванням, була знайдена лише одна пара ідентичних несправностей.

6. Висновки

На основі всього викладеного вище можна зробити висновок, що N -версійне програмування не має альтернативи щодо ефективності при виявленні несправностей проектування ПЗ гарантоздатних обчислювальних систем. Тому воно є потенційно ефективним методом забезпечення відмовостійкості ПЗ і заслуговує на подальші дослідження.

У той же час надзвичайно важливим методом для попередження активізації несправностей у бортових автономних системах довготривалого функціонування є метод прогнозування несправностей на основі імовірнісно-фізичних моделей відмов (дифузійних розподілів), запропонований фахівцями ІПММС НАНУ [11]. Перспективність розвитку цих моделей і широке застосування в інформаційних технологіях обумовлюються їх ефективністю. Витрати на проектування ПЗ, прогнозування несправностей та попередження відмов мають високу окупність у порівнянні з втратами часу та продуктивності ПЗ на виявлення помилки, оброблення несправності та відновлення після уже наявної помилки.

Крім того, розвиток методології проектування з використанням нових інструментальних засобів статистичного та імітаційного моделювання складних систем на основі мереж Петрі та Е-мереж [12] дає можливість створення, тестування і верифікації проекту бортового розподіленого комплексу, побудови моделі повної специфікації його функціонування та комплексного дослідження сценаріїв поведінки ще до його фізичної реалізації та експлуатації. Ці моделі оптимізують архітектуру, виявлення помилок проектування і можливість вбудови їх у контур керування

прогнозуючого елемента перевірки різноманітних гіпотез щодо майбутньої поведінки системи. Все це потребує подальших комплексних досліджень та імплементації у національній технології.

СПИСОК ЛІТЕРАТУРИ

1. Basic Concepts and Taxonomy of Dependable and Secure Computing / A. Avizienis, J.-C. Lapri, B. Randell et al. // IEEE Trans. On Dependable Secure Computing. – 2004.– Vol. 1, N 1. – P. 11 – 33.
2. Харченко В.С. Гарантоздатні системи та багатоверсійні обчислення: аспекти еволюції // Радіоелектронні і комп'ютерні системи. – 2009. – № 7 (41). – С. 46 – 59.
3. Дослідження відмовостійких обчислювальних засобів у критичних інформаційних системах обробки інформації та керування об'єктами на основі мережної взаємодії (шифр "Відмовостійкість") / Б.Г. Мудла, Г.С. Теслер, О.В. Федухін О.В. та ін. / Наук. звіт з фунд. досліджень ІПММС НАН України. – Державний обліковий № 0204U006760. – К., 2004. – 264 с.
4. Information Technology Security Evaluation Criteria, Harmonized criteria of France, Germany, the Netherlands, the United Kingdom, Commission of the European Communities. – 1991.
5. Pfleeger C.P. Data Security, Encyclopedia of Computer Science / A. Ralston et al., eds. – Nature Publishing Group. – 2000. – P. 504 – 507.
6. Laprie J.-C. Dependable Computing and Fault-Tolerance // Digest of Papers FTCS-15. – 1985. – June. – P. 2 – 11.
7. Randell B., Lee P.A., Treleaven P.C. Reliability Issues in Computing System Design // Computing Surveys. – 1978. – Vol.10, N 2. – P. 124 – 165.
8. Aviizienis A. Dependable Computing Depends on Structured Fault Tolerance // Sixth International Symposium on Software Reliability Engineering. – 1995. – P. 158 – 168.
9. Avizienis A. The N-Version Approach to Fault-Tolerant Software // IEEE Transactions on Software Engineering. – 1985. – SE-11(12). – P. 1491 – 1501.
10. Avizienis A., Lyu M.R., Shutz W. In search of effective diversity: a six-language study of fault-tolerant flight control software // Technical Report CSD-870060, UCLA Computer Science Department. – Los Angeles, California, 1987. – P. 15 – 22.
11. Стрельников В.П., Федухин А.В. Оценка и прогнозирование электронных элементов и систем. – К.: Логос, 2002. – 486 с.
12. Казимир В.В. Основні концепції побудови розподілених динамічних систем управління // Вісник Чернігівського технологічного інституту: Збірник. – Чернігів: ЧТІ, 1999. – № 9. – С. 145 – 152.

Стаття надійшла до редакції 22.06.2009