RADIO ENGINEERING, ELECTRONICS AND ELECTRICAL ENGINEERING

DOI 10.51582/interconf.7-8.04.2021.067

Цирульник Сергій Михайлович
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук та економічної кібернетики
Вінницький національний аграрний університет,
Вінницький технічний коледж, Україна

Бородай Ярослав Олександрович
викладач комп'ютерного відділення
Вінницький технічний коледж, Україна

Ткачук Василь Миколайович
викладач радіотехнічного відділення
Вінницький технічний коледж, Україна

Непийвода Максим Васильович
викладач комп'ютерного відділення
Вінницький технічний коледж, Україна

ПРОГРАМНЕ РЕЗЕРВУВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ

Анотація. *У статті розглядаються питання, що пов'язані зі стійкістю та надійністю програмного забезпечення мікропроцесорних систем. Наводиться аналіз статичного та динамічного резервування програмного забезпечення та рекомендації для безперервного виконання закладеної в мікропроцесорну систему керуючої програми.*
Ключові слова: *надійність, стан мікропроцесорної системи, стійкість програмного забезпечення, програмне статичне резервування, програмне динамічне резервування*

Надійність – властивість об'єкта зберігати в часі у встановлених межах значення всіх параметрів, що характеризують здатність виконувати необхідні

функції в заданих режимах і умовах застосування, технічного обслуговування, зберігання і транспортування. Таким чином, надійний об'єкт або система більшу частину часу повинна перебувати в справному або хоча б працездатному стані (рис. 1). Однак, зважаючи на всеосяжне розповсюдження складних мікропроцесорних систем у виробничих, в тому числі з підвищеною небезпекою технологічних процесах, напрямках контролю за критичними станами життєво важливих об’єктів, експертних системах прийняття рішень та надання доступу надійні системи не завжди є безпечними і тимчасова відмова чи збій в їх роботі можуть нести неприйнятно високі ризики та навіть призводити до катастрофічних наслідків.

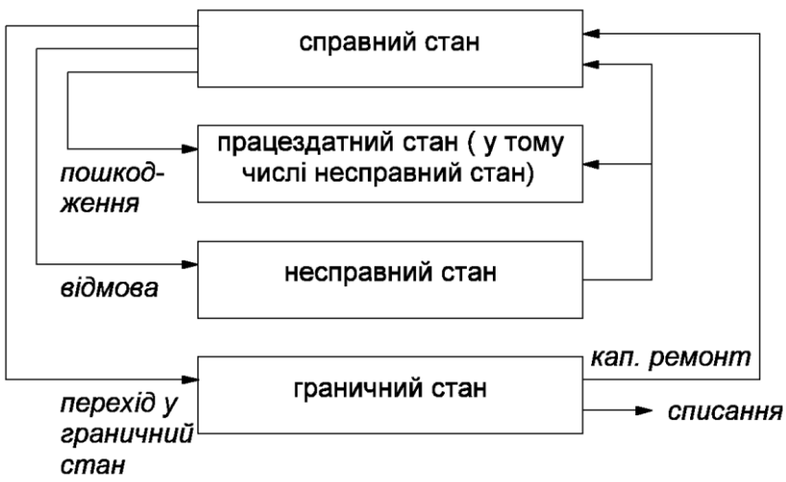

Рис. 1. **Стан системи з точки зору надійності**

Працездатність програмно-керованих компонентів систем забезпечується коректністю виконання закладеної програми, з одного боку, і безпомилковістю самого програмного забезпечення - з іншого. Якщо надійність електронних компонентів пов'язана з технологічним циклом їх використання і гарантується виробником (постачальником), то за безпомилковість програмного забезпечення і надійність системи як єдиного цілого повністю відповідає розробник.

Для критично важливих застосувань може знадобитися резервування апаратно-програмних комплексів. Резервування дозволяє значно підвищити

надійність системи в цілому при використанні менш надійних компонентів за рахунок введення надлишку апаратних чи програмних засобів [1].

Резервування може виконуватися динамічно:

– підключенням виділеної резервної підсистеми;

– виконанням критичних функцій справними підсистемами (перерозподіл навантаження);

або статично:

– мажоритарною обробкою сигналів від непарного числа паралельно однакових підсистем, що працюють;

– пріоритетною обробкою сигналів від довільного числа паралельно гетерогенних підсистем, що працюють.

При цьому основним завданням, яке вирішується резервуванням є забезпечення безперервного виконання закладеної в мікропроцесорну систему керуючої програми.

На відміну від апаратних засобів, що мають кінцевий ресурс експлуатації та схильні до збоїв, «термін придатності» програмного забезпечення необмежений. Тому надійність окремо взятої програми повністю закладається на етапі її розробки і включає два фактори: стійкість і коректність.

Під стійкістю програмного забезпечення розуміють зберігання його працездатності при відмовах апаратних засобів або помилки у вхідних даних. Наприклад, у разі виходу з ладу будь-якого давача (виявляється шляхом експертної оцінки кореляції даних давача із заданою множиною можливих їх значень) програмне забезпечення може використовувати інформацію з інших давачів або розрахункові дані, отримані заздалегідь шляхом попереднього моделювання системи.

Програмне резервування реалізовано, наприклад, в блоці управління двигуном (ECU) автомобіля. У разі відмови давача масової витрати повітря автомобіля необхідна для вприскування потрібної кількості палива величина витрати повітря розраховується паралельно за обертами двигуна та за положенням дросельної заслінки. В іншому випадку, при некоректному сигналі від давача температури охолоджуючої рідини, використовується

фіксоване значення 80°C, що відповідає температурі прогрітого двигуна. Крім того, ECU постійно запускає вентилятор охолодження, захищаючи двигун від можливого перегріву. Такі заходи дозволяють, в більшості випадків, самостійно доїхати до станції техобслуговування.

Для виявлення відмови може бути використана апріорна інформація про перебіг процесу і пов'язаних з ним даних. Функціональні перевірки можуть бути простими, наприклад виявлення постійного рівня сигналу протягом заданого інтервалу часу, тоді як за алгоритмом функціонування сигнал має змінюватися. Така ситуація інформує про обрив або замикання в електричних колах.

Більш складні перевірки передбачають порівняння даних, що надходять з поведінкою математичної моделі процесу в аналогічних умовах. Значна розбіжність також буде інформувати про несправності, причому ця ж математична модель може бути використана для подальшого програмного резервування (рис. 2).

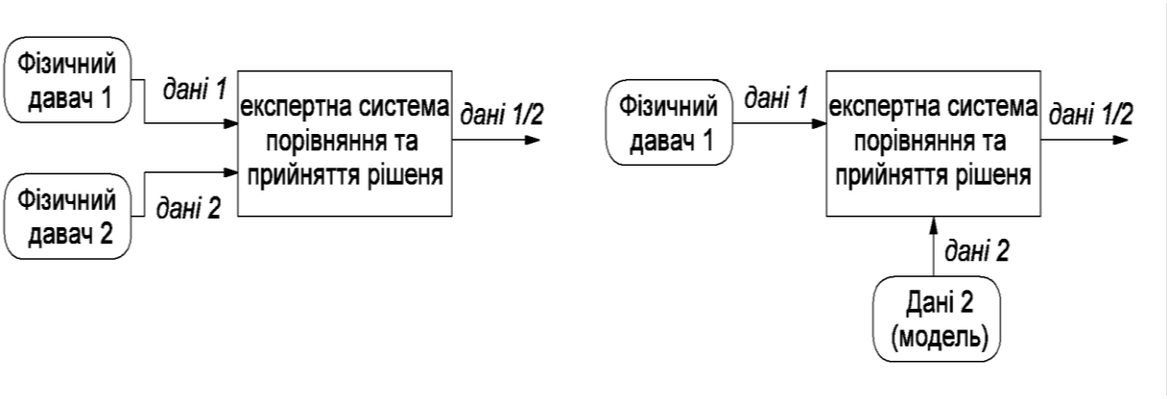

Рис. 2. **Апаратне (ліворуч) і програмне резервування**

Найбільшого розвитку програмне резервування набуло в концепції «цифрового двійника» (Digital Twin). Згідно з цією концепцією, ще на етапі розробки реального фізичного пристрою одночасно створюється його програмний аналог, в якому описані математичні моделі всіх складових частин та пристрою в цілому, а також алгоритми поведінки реального об'єкта в умовах впливів перешкод і навколишнього середовища [2].

У програму двійника впродовж життєвого циклу об’єкту, що контролюється, в режимі on-line заносяться дані з давачів, які розташовані на самому об’єкті, а також дані про оточуюче його середовище.

Така багатомірна системна модель породжує велику кількість додаткової інформації, яку необхідно генерувати, передавати та обробляти в короткі проміжки часу. Однак, вона дає можливість не тільки оптимізувати взаємодію складових частин пристрою з урахуванням режимів роботи та впливу навколишнього середовища, а що важливо з точки зору надійності, промоделювати різні варіанти повних і часткових відмов шляхом інтерактивної оцінки його технічного стану за сукупністю даних та перейти до ремонтно-відновлювальних робіт в розрахований термін.

За статистичними даними, описаний підхід дозволяє усунути до 70% аварій ще до їх виникнення [2].

Інші способи підвищення стійкості – програмна фільтрація вхідних даних, контроль та відновлення стану для компонентів з пам'яттю.

Коректність програмного забезпечення передбачає виконання всіх запланованих дій без побічних ефектів і визначається помилками, допущеними при його розробці. Помилки в програмному забезпеченні можуть виникати як з вини програміста (невідповідність вимогам замовника, вибір невідповідних алгоритмів, помилки реалізації мовою програмування, незнання або ігнорування особливостей мови та компілятора), так і внаслідок неправильної роботи інструментальних засобів (середовища розробки, компілятора, програматора) або готових бібліотек. Відповідно, основний шлях підвищення надійності програмного забезпечення – боротьба з помилками на всіх етапах розробки: уникання їх шляхом грамотного проектування і кодування, виправляння за допомогою цілеспрямованого пошуку на етапі налагодження та тестування, мінімізування їх наслідків на етапі експлуатації.

Складність сучасного програмного забезпечення не дозволяє гарантувати повну відсутність помилок навіть при застосуванні спеціалізованих інструментів статичного і динамічного тестування. Тому, за аналогією з апаратними засобами, підвищити надійність програмного забезпечення можна

шляхом введення надмірності – статичним або динамічним програмним резервуванням. Так, програмне статичне резервування реалізується шляхом паралельного виконання виконуваного коду подібної функціональності, але розробленого різними програмістами на різних платформах. Такий підхід називається багатоверсійним програмування (N-version programming, рис. 3)

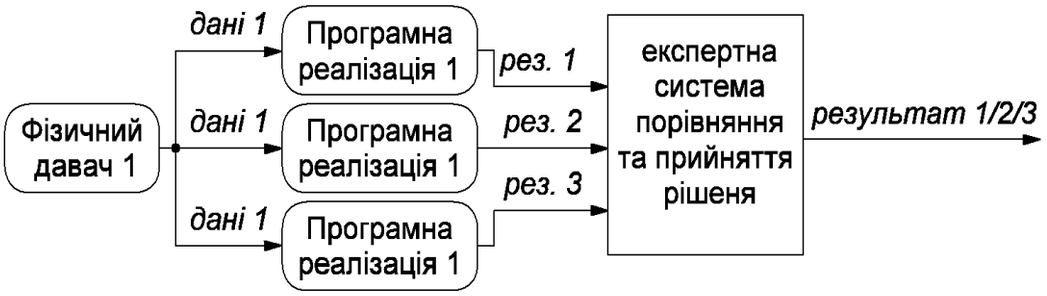

Рис. 3. **Резервування методом багатоверсійного програмування**

Фінансові витрати на реалізацію методу в умовах складності програмних продуктів можуть стати неприйнятними.

Модифікований підхід передбачає дублювання лише основних функцій системи із застосуванням більш простих та надійних методів з допустимим у разі відмови збитком для точності та універсальності. Основна функціональність системи забезпечується більш складним модулем, але в разі її несправності система перемикається на більш простий перевірений модуль (рис. 4).

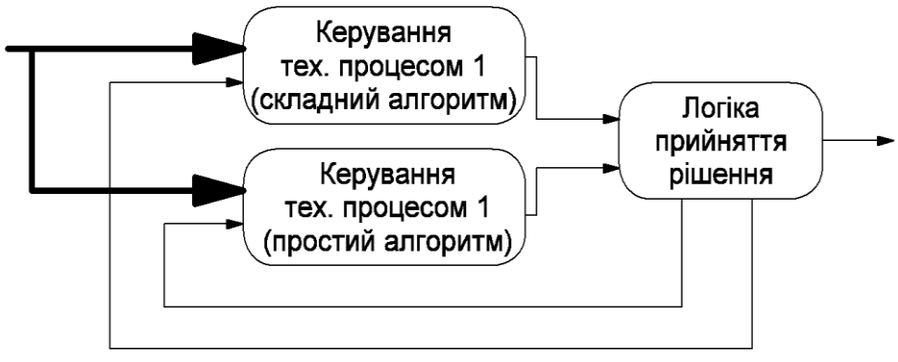

Рис. 4. **Резервування системи керування технологічним процесом**

Динамічне програмне резервування полягає в повторному виконанні тієї або нової альтернативної операції в разі помилки. Такий підхід вимагає менше ресурсів і може бути реалізований на рівні окремих функцій програми. При цьому необхідно виконати такі дії:

– виявити помилку;

– мінімізувати ефект помилки;

– відновити безпомилковий стан;

– показати збій (сигналізувати про збій) та прийняти подальше рішення.

Для виявлення помилкового стану можна скористатися формальним та функціональним контролем аргументів, перевіркою сигнатур. При цьому виявлене помилкове значення виправляється на найближче допустиме, середнє або останнє коректне значення. Для запобігання зациклення при очікуванні відгуку від несправного зовнішнього пристрою можна використовувати додаткові лічильники порожніх циклів або таймери, інтервал-рахунки яких відповідають найбільшому значенню затримки пристрою. Переповнення таймера до отримання результату (тайм-аут) інформує про помилку. Очікування з тайм-аутом реалізовано практично у всіх службових викликах багатозадачних систем.

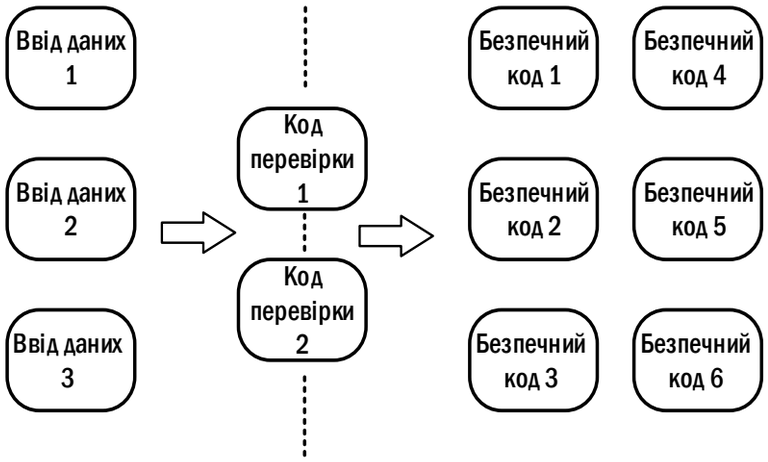

Рис. 5. **Виділення частини коду для перевірки даних**

Комплекс усіх цих заходів іноді називають захисним програмуванням. Ідея в тому, що створюється якась програмна «стіна» – шар контролюючих

процедур, які не допускають трансляції помилок з «брудної» частини програмного забезпечення в «чисту» надійну (рис. 5).

Помилки можуть проявлятися і як виняткова ситуація (exception): збій живлення (BOR), «холодний» запуск (POR), спрацьовування сторожового таймера (WDT), ділення на нуль. Оскільки для забезпечення надійності потрібно не тільки зафіксувати помилку, але і відновити працездатність, необхідний аналіз причин такого переривання. Для цього необхідна додаткова інформація, яку керуюча програма, або кожен з програмних модулів повинні час від часу записувати до енергонезалежної пам'яті. Аналіз динаміки зміни такої інформації проводиться експертною підсистемою та постійно верифікується із вбудованою базою даних – множиною вірогідних станів, після чого виноситься рішення щодо подальшої дії .

Експертні системи функціонують на основі нечіткої логіки (fuzzy logic) і дають можливість оперувати вхідними даними, які неможливо задати однозначно, або які змінюються в часі. У контексті надійності fuzzy-системи дозволяють реалізують алгоритми швидкого моделювання складних динамічних апаратно-програмних продуктів та проводити їхній порівняльний аналіз із заданим ступенем точності [3].

Прикладом застосування експертної системи в технологічному обладнанні є сучасна пральна машина із системою fuzzy logic, яка на основі аналізу даних від вбудованих давачів та бази даних вірогідних станів проводить не лише вибір робочих режимів (яких може бути декількох тисяч) та керує процесом прання, а й визначає відмови в роботі складових частин машини, помилки користувача, збої, що викликані впливом зовнішніх факторів (понаднормове відхилення напруги чи частоти в мережі живлення), і як результат – блокує роботу автомата в критичних випадках, що упереджує виникнення аварій.

Для відновлення безпомилкового стану системи найчастіше використовують механізм транзакцій, відповідно до якого поведінка системи представляється послідовністю коректних станів та переходів (транзакцій) між ними. При виявленні несправності відбувається повернення до останнього

коректного стану (rollback), для чого в процесі роботи зберігаються копія модифікованих даних і вся послідовність дій, що виконана в поточній транзакції. У разі безпомилкового завершення всіх необхідних дій транзакція підтверджується (commit), і система переходить до нового коректного стану. Подібний підхід роботи з даними реалізований в алгоритмі блокчейна (blockchain) та широко використовується в більшості сучасних систем управління базами даних [4].

Алгоритм транзакцій, чудово працює з даними, однак, його складно реалізувати при управлінні реальними процесами, які протікають безперервно і є незворотними. Відновлення попереднього стану системи не завжди можливо або пов'язано з порушенням вимог безпеки. Модифікований підхід, більш придатний для мікроконтролерів і передбачає виконання послідовності альтернативних дій в залежності від помилки і схожий з багатоверсійним програмуванням. При цьому блоки перевірок і відновлення оптимізуються під конкретний процес (рис. 6).

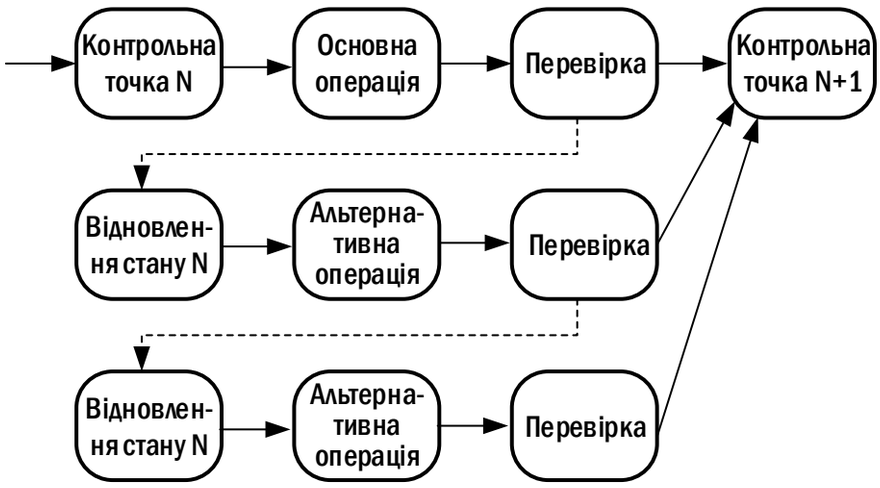

Рис. 6. **Блоки відновлення як приклад динамічного резервування**

Іноді для підвищення надійності та живучості доцільно застосовувати кілька мікроконтролерів, особливо якщо вони пов'язані з принципово різними реальними процесами. В цьому випадку модифікація програмного забезпечення одного мікроконтролера не зачепить програмного забезпечення іншого. Приклад відокремленої підсистеми - блок управління пасивною

безпекою автомобіля (SRS), що функціонує незалежно від блоку управління двигуном (ECU).

Зростання числа помилок та їх відновлення може бути передумовою до виходу з ладу певних компонентів системи, які можна своєчасно попередити.

Поєднання засобів візуалізації етапів технологічного процесу із функціями керування ним реалізоване в системах СКАДА (SCADA), які представляють собою мікропроцесорну систему призначений для збирання (в більшості дистанційного), обробки, відображення та збереження інформації про фізичний об'єкт, а також його управління в режимі реального часу [5].

Функціонування таких систем передбачає наявність диспетчера, який відіграє головну роль в прийнятті рішень, однак СКАДА проводить неперервну регенерацію баз даних, аналіз поточного стану технологічної системи, що контролюється, який фіксується (відображається) за допомогою графічного інтерфейсу. Крім того СКАДА також генерує тривожні сигнали та повідомлення про аварійні випадки, архівує історію роботи контрольованої системи та оператора, здійснює постійний контроль над його діями чим мінімізує навмисний чи ненавмисний вплив на безпечне та безперебійне функціонування технологічної системи людського чинника.

Висновки. Використання сучасних складних мікропроцесорних систем для збирання та обробки первинної інформації, збереження та перетворення даних, прийняття рішень та генерації керуючих сигналів в технологічних застосуваннях має незаперечні переваги, однак несе й ризики пов’язані із відмовами, як апаратних так і програмних складових зазначених систем, що може призводити навіть до катастрофічних наслідків.

Питання надійності функціонування та мінімізації ризикових наслідків застосування складних керуючих програмно-апаратних продуктів на зараз у загальному відпрацьовані та реалізовані у вигляді окремих ефективних апаратно-програмних інструментів, які базуються на різних алгоритмах, однак не мають закінченого «універсального» теоретичного підґрунтя.

Накопичення, структурування, обробка та збереження великих надлишкових масивів інформації за принципами Big Data з елементами

штучного інтелекту та нечіткої логіки, новітніх алгоритмів збереження достовірних даних, які отримали розвиток в останньому десятиріччі, очевидно, є базовими інструментами комплексного вирішення питань підвищення надійності функціонування складних керуючих апаратно-програмних продуктів мікропроцесорних систем.

Список джерел:

1. Солодов В. С., Калитёнков Н. В. Надежность радиоэлектронного оборудования и средств автоматики. СПб.: Издательство «Лань», 2018. 220 с.
2. Цифровой двойник – CADFEM CIS. URL: https://www.cadfem-cis.ru/service/digital-twin.
3. Нечеткая логика в системах управления. URL: http://csef.ru/ru/articles/print/554.
4. Мащенко П. Л., Пилипенко М. О. Технология Блокчейн и ее практическое применение. Наука, техника и образование. 2017, 2 (32). С. 61-64.
5. Щагин А. В., Демкин В. И., Кононов В. Ю., Кабанова А. Б. Основы автоматизации технологических процессов. М.: Издательство Юрайт, 2019. 163 с.