УДК 628.852: 53.08 + 004.73

Кубов В. І., Зюляєв Д. Д.

ЕВРИСТИЧНІ АЛГОРИТМИ САМОВІДНОВЛЕННЯ ПРАЦЕЗДАТНОСТІ МІКРОКОНТРОЛЕРНОЇ СИСТЕМИ ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ

Рассматриваются проблемы построение систем сбора и обработки информации на базе программного обеспечения с негарантированной надежностью.
Ключевые слова: *программное обеспечение, негарантированная надёжность.*

Розглядаються проблеми побудови системи збора та обробки інформації на базі програмного забезпечення з негарантованою надійністю.
Ключові слова: *програмне забезпечення, негарантована надійність.*

This article describe problem building system out-in data based on program software with unpredictable robustness.
Key words: *software, unpredictable robustness.*

Огляд загальних проблем розробки відмовостійкого програмного забезпечення для мікроконтролерних систем. Для реалізації ефективних систем керування енергозбереженням потрібно мати постійний контроль за поточними параметрами системи. Для контролю параметрів сучасних систем традиційно використовують мікроконтролерні засоби.

Елементи мікроконтролерної системи отримують інформацію з сенсорів в вигляді електричних параметрів: напруга, струм, частота електричних коливань або інше. Ці елементи не мають ніяких посередників і являються першою ланкою у ланцюгу послідовних етапів обробки інформації. Зважаючи на вимогу низької вартості системи, як правило, використовуються найпростіші мікроконтролери. Особливістю таких мікроконтролерів є малий обсяг пам’яті та обмеження швидкодії, що вимагає особливої ретельності при розробці програмного забезпечення та його валідації.

Практикою сучасної розробки програмного забезпечення є широке використання готових бібліотек та процедур для реалізації складних елементів протоколів обміну між окремими складовими системи. Ці бібліотеки з точки зору розробника програмного забезпечення виглядають, як «чорна скринька» з певними умовами на вході та виході. Очікувати адекватних результатів можна лише за умов відповідності апаратного та програмного середовища розробника бібліотеки та користувача цієї бібліотеки. Але повна відповідність апартно-програмного забезпечення можлива лише при повному дублюванні апаратури та алгоритмів обробки, тобто при звичайному тиражуванні існуючої розробки.

Якщо ставиться мета розробки нового оригінального продукту, то умови ідентичності апаратного та алгоритмічного середовища порушуються. Це спричиняє «незрозумілі», раптові відкази окремих елементів комплексу або всього комплексу. Для боротьби з такими відказами можна намагатися використовувати «чесний метод» – пошук витоків проблеми – bugs, шляхом аналізу текстів алгоритмів бібліотек, та усунення виявлених помилок. Але такий метод вимагає глибокого занурення у проблематику відповідної задачі з надмірно великими витратами часу. Окрім того такий підхід не завжди можливий, через поширену практику використання модулів «інтелектуальної власності» з відсутніми текстами початкових програмних кодів. Тому при розробці нового програмного забезпечення вимушено обмежуються умовами, коли помилки залишаються і інколи виникають, але не дуже часто. За таких умов надійність роботи системи набуває ймовірнісного характеру.

Якщо заздалегідь є припущення про можливий відказ працездатності системи, то необхідно передбачити методи найшвидкішого відновлення системи з мінімально припустимими втратами інформації.

Система повинна відновлюватися навіть за умов непередбачених заздалегідь (на етапі проектування програмного забезпечення) помилок або збоїв. Невизначеність помилкових умов та засобів усунення наслідків помилок примушує використовувати евристичні методи. Назва евристичний походить від грецького εὑρίσκω (heuristiko) – знахідка[1].

Проблеми побудови надійних систем з ненадійних компонентів давно відома і достатньо успішно розв’язується при розробці великих проектів на базі відкритого програмного середовища Linux [2]. В закордонній термінології до програмного забезпечення, що відповідає умовам надійності використовується термін –

Robustness Software. В вітчизняній практиці термін «робастний» використовується переважно для позначення стійкості алгоритму до суттєвих помилок вхідних даних. Але треба розглядати стійкість системи значно ширше: не тільки відносно помилок даних, але й відносно власних помилок алгоритму обробки, та комплексних помилок, як програмних так і апаратних, комплексу в цілому.

Проблеми реалізації системи контролю температурних режимів університету

З задачею підвищення надійністю системи автори зіткнулися під час розробки системи моніторингу температурних режимів університету[3]. Було розроблено систему для отримання температурних даних з мікроконтролерних веб-серверів [4]. Спрощену схему мережі наведено на рис. 1.

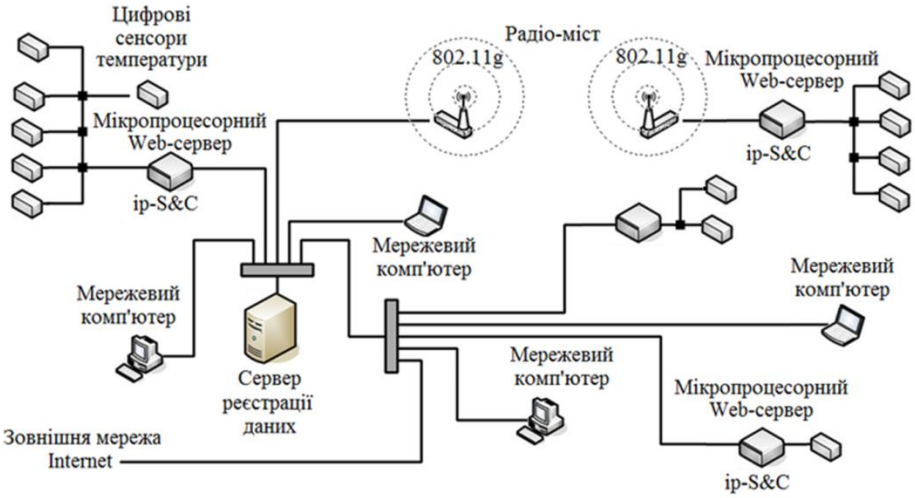


Рис. 1. Схема підключення мікропроцесорних Web-Серверів до локальної комп'ютерної мережі університету.

Сервер збору даних фігурує як клієнта для Web-Сервера й із заданою періодичністю (приблизно раз у хвилину) опитує всі контрольовані точки. Результати вимірів фіксуються на жорсткому диску сервера збору даних, і далі можуть бути оброблені й перетворені в графіки в пакеті Excel за допомогою спеціально розроблених макросів [5].

Такі данні отриманні на інтервалі 2011-2013 роки з темпом близько 5 хвилин по кожному з 16 температурних сенсорів (понад 4 млн. окремих відліків). Приклад результатів реєстрації температур з кількох сенсорів, розташованих у різних приміщеннях і спорудах університету протягом однієї доби наведено на рис. 2.

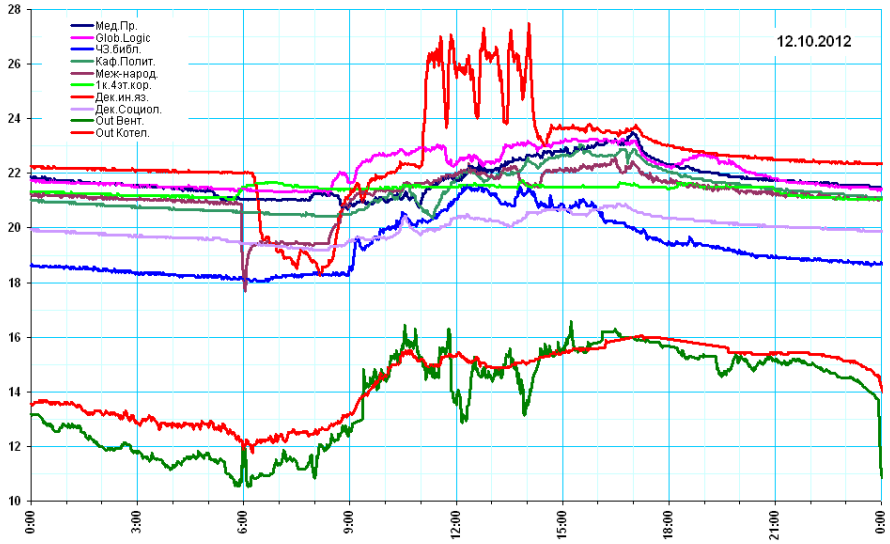


Рис. 2. Приклад варіацій температури в приміщеннях університету і температури зовнішнього повітря протягом однієї доби.

Досвід налагодження системи отримання даних температурних режимів університетів являє яскравий приклад підвищення надійності системи шляхом використання евристичних методів.

В якості прототипу програмного забезпечення для мікроконтролерного веб-серверу збору даних було використано відкриті програмні коди для організації мережевого зв'язку між мікроконтролером та Ethernet-мережею. Примірники програми обміну з Ethernet-мережею не мали ні яких ознак непрацездатності при ручному тестуванні.

Впровадження цих процедур у автоматичну систему після кількох хвилин нормальної роботи призвело до раптової зупинки. Як пізніше з'ясувалося, причиною

зупинки була помилка ініціалізації змінної програми, що приводила до переповнення стеку після кількох десятків звернень до сенсорів. Ця помилка була виправлена, але залишилась невпевненість в відсутності в програмі інших, подібних помилок.

Як було встановлено при експериментах з елементами системи, відмова одного мікроконтролерного сервера відбувається орієнтовно не частіше одного разу на два тижні. Відповідна ймовірність добового відмови $\overline{P_1(day)} \approx 1/14$. З урахуванням типової частоти запитів, а саме 1 запит до 5 хвилин, отримаємо ймовірність відмови на запит

$$\overline{P_1(\text{request})} \approx \frac{1}{14 \cdot 24 \cdot 12} = 1/4032 \approx 0.25 \cdot 10^{-4}.$$

Виявлення причини відмови при такій низькій вірогідності є надзвичайно складним завданням, та вимагає значних витрат часу.

Разом з тим, низька ймовірність відмови одного сервера перетворюється у високу ймовірність відмови хоча б одного сервера при великій кількості серверів. Для N серверів ймовірність відмови системи визначається рівнянням

$$\overline{P_N(day)} = (1 - P_N(day)),$$

де $P_N(day) = P_1(day)^N$ – ймовірність безвідмовної роботи системи з N елементів. Ця ймовірність пов'язана з ймовірністю безвідмовної роботи одного елемента системи $P_1(day) = 1 - \overline{P_1(day)}$.

Таким чином ймовірність відмови системи з N елементів визначається рівнянням

$$\overline{P_N(day)} = 1 - \left(1 - \overline{P_1(day)}\right)^N.$$

Підставляючи конкретні значення ймовірності відмови для експериментальної системи, що складається з 10 серверів, отримаємо

$$\overline{P_{10}(day)} = 1 - \left(1 - \overline{P_1(day)}\right)^{10} = 1 - (1 - 1/14)^{10} \approx 1 - 0.47 = 0.53$$

Тобто при такій ймовірності відмови хоча б в одному елементі системи повинні відбуватися приблизно один раз на день – раз на два дні.

Саме така ситуація, на жаль, і спостерігалася в експериментальній системі. Відмови одного мікроконтролера відбувалися дуже рідко, щоб можна було виявити причину відмови, але занадто часто для системи в цілому.

Для усунення наслідків відмови – «підвисання» мікроконтролера, доводилося підходити до місця розміщення конкретного контролер і виконувати операцію його апаратного скидання.

Апаратно-програмні засоби самовідновлення роботи елементів системи. Для можливості автоматичного самовідновлення системи побудованій на мікроконтролері більшість виробників передбачають, так званий, механізм *Watchdog – Сторжевий Пес*. Головна ідея цього механізму полягає в тому, що програма повинна періодично відзначати свою працездатність записом в спеціальний регістр[6]. Якщо такого запису не було зроблено, то спрацьовує таймер примусового пере запуску мікроконтролера.

Використання механізму Watchdog має певні особливості. А саме:

– Максимальний інтервал часу між відзначками працездатності не повинен перевищувати певного максимального значення. Тому треба враховувати часові проміжки при виконанні певних процедур в програмі.

– Деякі бібліотечні елементи програми можуть затримувати виконання програми на час, що перевищує максимально припустиме значення. Отже таймер повинен скидатися при початку виконання процедури та по закінченню.

В програмі мікроконтролерного веб-серверу Watchdog реалізовано в двох місцях: перше при ініціалізації протоколу обміну, для впевненості, що всі компоненти справні; друге при отриманні, обробки та відсиланні даних в разі затримки виконання програмного коду, або не отримання запиту виконується перезапуск серверу.

Щоб автоматизувати процес перезапуску до програмного коду мікроконтролерного веб-серверу додано процедуру Watchdog, яка слідкує за станом серверу і у разі не отримання запиту через 5 хвилин примусово перезавантажує програму. Але дана процедура є лише закриває проблему і не вирішує її.

Фрагмент програми обробки запитів

```
void loop(){
uint16_t dat_p;
delay(10); // затримка на 10 мс
counter++; // додання лічильника
if (counter<=30000) wdt_reset();  // перезапуск при досягнені ліміту таймера
dat_p=es.ES_packetloop_icmp_tcp(buf,es.ES_enc28j60PacketReceive(BUFFER_SIZE,buf));
if(dat_p!=0){ // наявність запиту в мережі
if (strncmp("/ ",(char *)&(buf[dat_p+4]),2)==0){ // перевірка правильності запиту
dat_p=print_webpage(buf); // обробка запиту
}
es.ES_www_server_reply(buf,dat_p); // якщо запит не правильний, то пропустити
}
}
```

Процедура обробки запитів

1. Ініціалізуем лічильник та робимо паузу – 10 мс
2. Збільшуємо лічильник на одиницю
3. Перевіряємо чи не досяг лічильник максимального значення – 30000, час очікування – 5 хвилин (10ms*30000=300s=5m), виконуємо перезапуск.

4. Якщо є запит з мережі
1.1. Якщо правильний запит:
4.1.1. Формуємо відповідь
4.1.2. Скидаємо лічильник
2.2. Відсилаємо данні
5. Повторюємо всі операції починаючи с п.1.

В подальшому тестуванні виявлено, що інколи при перезапуску мікроконтролерний сервер та сервер збору можуть попасти в один тайм-слот і призвести до спотворення даних. Тому в логах даних іноді трапляються неправдиві данні з частотою один – три рази на день.

Одним з неочікуваних джерел помилок виявився механізм початкового завантаження програми у пам'ять процесора, так званий Bootloader. Цей механізм було розроблено з метою полегшення завантаження програм. Недоліків у будь-якого бутлоадера два - найголовніший використовується частина флеш пам'яті. Другий, не менш вагомий, то що він стартує перший і якщо не подбати про правильний алгоритм входу в бут, то мікроконтролер може зробити це мимовільно, записавши в себе неправильний код [7].

З'ясувалось, що полегшений механізм завантаження, полегшує і умови для саморуйнування програми під час частих перезапусків в наслідок відмов системи електропостачання. При неправильному виключенні живлення руйнувався програмний код. Тому для виключення можливості само руйнування програмного коду за допомогою налаштувань FUSE відключено завантаження бутлоадера.

Досвід експлуатації комплексу показав, що навіть після усунення потенційних джерел помилок, якісь невідомі помилки залишилися. Ці помилки приводили до раптової відмови в відповіді на запит від серверу і зависання мікроконтролера. Тож задача підвищення надійності роботи системи залишилась.

Як уже відзначалось, для такого класу задач добре підходять евристичні методи. Евристичний алгоритм спроможний видати прийнятне рішення проблеми серед багатьох рішень. Зазвичай такі алгоритми знаходять рішення близьке до найкращого і роблять це швидко. Такий алгоритм може бути точним, тобто знаходити найкраще рішення, але він все одно буде називатися евристичним, доти доки не буде доведено, що рішення дійсно найкраще.

Використання евристичних алгоритмів дає змогу накопичувати базу помилок знайдених у коді програми, що спрощує роботу програміста. Також можливе тимчасове усунення помилки застосовуючи різні процедури, які б запобігали виникненню не штатних ситуацій. Таким чином зменшуються витрати на усунення помилок при розробці, враховуючи, що більшість сучасного програмного забезпечення не виключає можливість збою.

Розглянемо варіант коли сервер отримання даних, припинив зчитувати данні, можливі причини:

1. Відсутність живлення - А
2. Відсутність локальної мережі - Б
3. Помилка при виконанні скрипту - В
4. Помилка в електронних елементів – Г

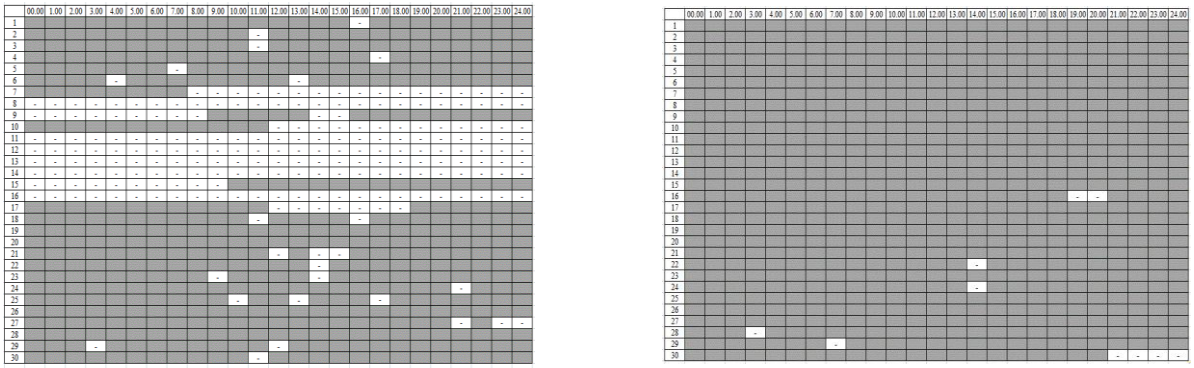
Вважаємо, що ймовірність помилок В та Г по 31 %, а А та Б по 18 %. 2% – це помилки, які можуть не враховуватися при написанні алгоритму самовідновлення.

При роботі евристичний алгоритм скоріш за все протестує найменш ймовірні проблеми А та Б, при негативному результаті буде розглянуто логи на факт помилки, яка частіше за все виникала В або Г для вирішення питання про метод усунення помилки. У разі збереження періодичних збоїв буде розглянута помилка, яка не ввійшла в жоден з переліків, а отже буде додана до списку.

Використання такого типу алгоритмів забезпечує зменшення ризику виникнення помилок, а також збільшує надійність та інформативність для розробника програмного забезпечення вказуючи на помилки, які виникли в ході експлантації системи.

Для вирішення проблеми зависання мікроконтролера в глобальному циклі програми відпрацьовується таймер перезапуску. При старті програми вмикається Watchdog та лічильник, який робить затримку на 10 мс потім скидається таймер Watchdog і починається процедура очікування запиту. Якщо запиту немає на протязі 5 хвилин відліку таймера мікроконтролер перезавантажується; у разі наявності запиту виконується процедура обробки-відправлення даних, а також в сей час таймер слідкує за станом виконання процедури і у разі перевищення виконання більш ніж дві секунди мікроконтролер перезавантажується.

В результаті впровадження алгоритмів само відновлення в рази зменшилися випадки зависання мікроконтролерного веб-серверу, а отже зменшилась кількість втрачених даних.



А *В*

Рис. 3. Приклади узагальненої статистики наявності даних
А – без алгоритму, В – з алгоритмом

Використання евристичних алгоритмів зменшує кількість можливостей спотворення або відсутності даних. Але треба зважати, що використання такого типу алгоритмів лише маскує проблему і дає відомості розробнику про неї. В свою чергу розробник повинен відшукати причину виникнення конфлікту у системі, але вже як зазначалося більшість розробників не мають можливості та ресурсів для пошуку всіх помилок та гарантування сто процентної відказо стійкості, отже використання евристичних алгоритмів являє собою економічно вигідний крок при розробці програмного коду.

Висновки та рекомендації

Під час експлуатації комплексу було виявлено ряд помилок, які призводили до відмов тому треба розробити систему яка:

1. Записує логи всіх помилок.
2. Періодичність виникнення помилок
3. Відслідковує запити, які надходять до системи

Для даного типу задач використання евристичних алгоритмів є досить доцільним, так як вони спроможні аналізувати стан системи та приймати рішення для підтримки працездатності системи. Це також зменшує витрати на системи, які будуть знаходити помилки, так як сам алгоритм в змозі виконувати пошук причини відмов працездатності. Представлені дані показали, що при використанні алгоритмів процент відмови знизився до двох процентів. Результати роботи можуть бути застосовані в галузях, де є задача виміру, отримання, первинної обробки та передачі даних, за умов відсутності 100% надійності апаратних та програмних засобів, що використовується системою. Використання відповідних апаратних та програмних засобів, а саме евристичних алгоритмів оцінки та відновлення працездатності системи дозволять підвищити надійність та ефективність роботи системи в цілому.

ЛІТЕРАТУРА

1. Эвристический_алгоритм. – Wikipedia, 2012. – [Цит. 2013, 15 квітень]. – Доступний з: < http://ru.wikipedia.org/wiki/Эвристический_алгоритм >.
2. Robust Definition, 2013. – [Цит. 2013, 18 квітень]. – Доступний з: http://www.linfo.org/robust.html.
3. Держ. реєстр. № 0113U001702
4. Андреев В. И., Дмитренко Н. В., Зюляев Д. Д., Кубов В. И., Черемисина В. В., Чухлебов А. В. Комплексная система учета и контроля температурных режимов административных зданий с помощью ip s&c системы и сети ethernet. \ Ольвійський форум 2012. Миколаїв : ЧДУ ім. Петра .Могили. 2012.
5. Дмитренко Д.В. Обробка температурних даних в системі «Енергоефективний університет». \ Наукові праці. т.169, вип.157. Техногенна безпека Миколаїв : ЧДУ ім. Петра Могили. 2011, с. 103–109.
6. Atmega 8 Datasheet, 2013. – [Цит. 2013, 20 квітень]. – Доступний з: http://www.atmel.com/Images/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf.
7. AVR. Учебный Курс. Использование Bootloader. – Сообщество easyelectronics.ru, 2013. – [Цит. 2013, 7 травня]. – Доступний з: http://easyelectronics.ru/avr-uchebnyj-kurs-ispolzovanie-bootloadera.html.

Рецензенти: ***Кутковецький В. Я.,*** д.т.н., професор;
Мусієнко М. П., д.т.н., професор.

Дата надходження статті до редколегії 14.05.2013 р.

КУБОВ Володимир Ілліч – к.ф.-м.н., доцент кафедри медичних приладів і систем, ЧДУ імені Петра Могили.
Коло наукових інтересів: електронні прилади для контролю середовища та біофізичних параметрів.

ЗЮЛЯЄВ Данило Дмитрович – аспірант кафедри екології та природокористування ЧДУ імені Петра Могили.
Коло наукових інтересів: системи збору і обробки даних.