МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Завідувач кафедри
_____ Скарга-Бандурова І.С.
«____»_____ 20__ р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

Дослідження методів підвищення відмовостійкості програмного забезпечення промислового контролера

Освітньо-кваліфікаційний рівень "Магістр"
Спеціальність 123 – «Комп'ютерна інженерія»

Науковий керівник роботи:	_____ (підпис)	Ларгін В.А. (ініціали, прізвище)
Консультант з охорони праці:	_____ (підпис)	Критська Я.О. (ініціали, прізвище)
Студент:	_____ (підпис)	Бережний А.Г. (ініціали, прізвище)
Група:		КІ-17дм

Сєвєродонецьк 2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерних наук та інженерії
Освітньо-кваліфікаційний рівень магістр
Напрям підготовки _____
(шифр і назва)
Спеціальність 123 – «Комп'ютерна інженерія»
(шифр і назва)

ЗАТВЕРДЖУЮ:

Завідувач кафедри _____
_____ І.С. Скарга-Бандурова
«_____» _____20____р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Бережному Анатолію Геннадійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів підвищення відмовостійкості програмного забезпечення промислового контролера

керівник проекту (роботи) Ларгін Віктор Анатолійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " " 2019 р. № ____
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз задачі дослідження
2. Проектування диспетчера операційної системи реального часу
3. Розробка диспетчера операційної системи реального часу
4. Охорона праці

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	Критська Я.О.		

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Пошук і дослідження літератури по темі дипломної роботи	2.07.2018 – 3.08.2018	
2	Підбір, вивчення та опрацювання практичних матеріалів на досліджуваному підприємстві	6.08.2018 – 17.08.2018	
3	Складання плану роботи	20.08.2018 – 31.08.2018	
4	Розробка та узгодження з керівником першого розділу диплома	3.09.2018-5.10.2018	
5	Розробка та узгодження з керівником другого розділу диплома	8.10.2018 -26.10.2018	
6	Розробка та узгодження з керівником третього розділу диплома	29.10.2018 – 14.12.2018	
7	Узгодження з керівником введення, висновків і пропозицій	17.12.2018 - 28.12.2018	
8	**Подання дипломної роботи на кафедру**	**9.01.2019**	
9	Підготовка доповіді та графічного матеріалу, ознайомлення з рецензією	10.01.2019 – 15.01.2019	
10	Захист випускної кваліфікаційної роботи	16.01.2019	

Студент _____ **Бережний А.Г.**
(підпис) (прізвище та ініціали)

Науковий керівник _____ **Ларгін В.А.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бережний А.Г. Дослідження методів підвищення відмовостійкості програмного забезпечення промислового контролера.

Удосконалено модель диспетчера керуючого обчислювального процесу шляхом дослідження організації паралельних обчислювальних процесів у мультиядерних мікропроцесорах, що дозволяє підвищити загальну відмовостійкість і зменшити витрати при впровадження її у систему. Практичне значення отриманих результатів полягає в тому, що основні наукові положення магістерської роботи реалізованих у виді моделі та програмних засобів трьохверсійного керуючого обчислювального процесу використано для подальшого проектування.

Ключові слова: відмовостійкість, мажорірованіє, мультиверсійність, диспетчер, програмне забезпечення

АННОТАЦИЯ

Бережной А. Исследование методов повышения отказоустойчивости программного обеспечения промышленного контроллера.

Усовершенствована модель диспетчера управляющего вычислительного процесса путем исследования организации параллельных вычислительных процессов в мультиядерных микропроцессорах, что позволяет повысить общую отказоустойчивость и уменьшить затрати при внедрении ее в систему. Практическое значение полученных результатов заключается в том, что основные научные положения магистерской работы, реализованные в виде модели и программных средств трёхверсионного управляющего вычислительного процесса, использованы для дальнейшего проектирования.

Ключевые слова: отказоустойчивость, мажорирование, мультиверсийнисть, диспетчер, программное обеспечение

ANNOTATION

Berezhny A.G. Investigation of methods of increasing the fail-safe reliability of the industrial controller software.

The model of the controller of the computing process is improved by studying the organization of parallel computing processes in multicore microprocessors, which allows to increase the overall fault tolerance and reduce the cost of its introduction into the system. The practical significance of the results obtained is that the main scientific positions of the master's work implemented in the form of a model and software tools of the three-conversion control computing process are used for further design.

Keywords: fault tolerance, majorization, multiverse, dispatcher, software

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – Програмне забезпечення;

СП – Спеціалізовані процесори;

ФБ – функціональний блок;

ОС – операційна система;

ОСРЧ – операційна система реального часу.

ВСТУП

Актуальність теми. Однією з основних проблем при розробці систем управління критичними об'єктами є створення високонадійного програмного забезпечення (ПЗ) керуючих контролерів [1]. При розробці ПЗ повинні вирішуватися два основні завдання:

– створення надійного (безпомилкового) базового програмного забезпечення;

– проектування і включення програмних засобів для забезпечення живучості при програмних, інформаційних і апаратних відмовах.

Відомим методом підвищення достовірності розроблення ПЗ є застосування об'єктно-орієнтованого і структурного підходів до програмування, перш за все за рахунок зменшення числа проектних помилок. При налагодженні написаних програм важливо ретельно провести їх тестування [6]. Повне тестування програми зазвичай нездійсненно на практиці, а експериментальне тестування програм може бути доказом наявності помилок, але ніколи не доведе їх відсутності. Тому підвищення достовірності обчислювального процесу з урахуванням можливих прихованих помилок в програмі є актуальним завданням.

Незважаючи на значний внесок в розвиток теорії і практики відмовостійкості керуючих систем, проблема практичної реалізації трьохверсійного обчислювального процесу вимагає подальших досліджень з урахуванням додаткових можливостей, що надаються новими розробками електронних компонентів і інформаційних технологій.

Суттєвою проблемою практичної реалізації трьохверсійного обчислювального процесу є налагодження і впровадження нових програмних модулів в реальні системи управління.

Найбільш імовірним джерелом систематичних відмов систем управління є дефекти програмних засобів, внесені при розробці, що не виявлені при тестуванні і верифікації і які з’являються при певному наборі вхідних даних або характеристиках фізичних або інформаційних середовищах. Резервування системи з виконанням ідентичних програмних додатків не вирішує проблему усунення таких дефектів, так як вони тиражуються і з’являються одночасно, викликаючи фатальну відмову.

Метою даної роботи є дослідження способів організації надійного відмовостійкого обчислювального процесу в керуючому промисловому контролері і практична реалізація мультиверсійного програмування.

Об'єкт дослідження – програмне забезпечення промислового контролера.

Предмет дослідження – методи реалізації трьохверсійності в мультиядерному контролері для підвищення відмовостійкості керуючих систем.

Проведені в роботі дослідження основані на теорії графів, марковських ланцюгів, теорії ймовірності та комбінаторики які використовувались при розробленні трьохверсійного керуючого обчислювального процесу.

В даний час в системах управління широкого поширення набули контролери з мікропроцесорами фірми Intel. Поява високошвидкісних багатоядерних мікропроцесорів (англ.

Multi-core) створило передумови для практичної реалізації мультиверсійного обчислювального процесу за рахунок апаратної та тимчасової надмірності.

Удосконалено модель диспетчера керуючого обчислювального процесу шляхом дослідження організації паралельних обчислювальних процесів у мультиядерних мікропроцесорах, що дозволяє підвищити загальну відмовостійкість і зменшити витрати при впровадження її у систему.

Проблема відмовостійкості в керуючих система розглядалися в таких роботах: R.T. Wood [11] - розглянуто використання диверсійності у ядерній промисловості, В.С. Харченко [16-18] - проаналізовано моделі багатоверсійних інформаційно-управляючих систем, М.А. Ястребенецького [19], В.В. Скляра [20] - переглянуто можливі способи адаптації у багатоверсійних мажоритарно-резервованих комп'ютерних системах управління, А.В Волкова [21], Ю.Н. Соколова [22] - проаналізовано метод оцінки надійності програмно-технічних комплексів, О.В. Брежнєва [23] - розглянуто метод оцінки безпеки критичної енергетичної інфраструктури, А.В. Федухіна [24], Н.В. Якимця [25] - розглянуто моделі проектування відмовостійких цифрових систем та ін.

Незважаючи на значний внесок в розвиток теорії і практики відмовостійкості керуючих систем, проблема підвищення достовірності розроблювального ПЗ вимагає подальших досліджень з урахуванням додаткових можливостей, що надаються новими розробками електронних компонентів і інформаційних технологій.

Практичне значення отриманих результатів полягає в тому, що основні наукові положення магістерької роботи реалізованих у виді моделі та програмних засобів трьохверсійного керуючого обчислювального процесу використано щодо розглядання, для подальшого проектування.

Результатом роботи є розроблене ПЗ - диспетчер для забеспечення трьохверсійного програмування на базі багатоядерного процесора.

Особистий внесок здобувача. Усі основні положення, результати, висновки та рекомендації магістерської роботиотримано автором самостійно.

Публікації. За темою магістерської роботи з викладенням її основних результатів опубліковано наукову статтю та опубліковано тези доповіді в двох збірниках наукових праць.

Структура та обсяг магістерської роботи. Магістерська робота складається із вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Загальний обсяг магістерської роботи складає 85 сторінок, з яких основний текст на 61 сторінках, список використаних джерел із 56 найменувань на 4 сторінках, додатки на 11 сторінках. Робота містить 2 таблиці, 2 додатки, 18 рисунки та 12 слайдів презентації.

1. АНАЛІЗ ЗАДАЧІ ДОСЛІДЖЕННЯ

Даний розділ присвячується теоретико-методологічним аспектам дипломної роботи. В даному розділі проаналізовано методи забезпечення відмовостійкості, розглянуті основні принципи побудови систем забезпечення безпеки шляхом мажорірованія, особливості мультиверсійних систем. На підставі аналізу сформована постановка задачі та розроблено технічне завдання.

1.1 Аналіз вимог для створення високонадійного програмного забезпечення

Однією з основних проблем при розробці систем управління критичними об'єктами є створення високонадійного програмного забезпечення (ПЗ) керуючих контролерів [1].

Вимоги високої надійності насамперед стосуються систем, критичних до безпеки (так званих safety-critical systems). До таких систем, зокрема, належать системи, які використовують в аерокосмічній і медичній галузях, в ядерній енергетиці, виробництві сучасної зброї, управлінні залізницями, в автомобільній промисловості, телекомунікаційній і мікропроцесорній індустрії та ін. Відмови в цих системах пов’язані із загрозою життю людей, втратою чи руйнуванням устаткування, забрудненням навколишнього середовища тощо. Процес розроблення високонадійних систем оснований на принципі зменшення ризику або повного виключення неспрацьовування умови безпеки в роботі системи.

Надійність — властивість технічних об'єктів зберігати у часі в установлених межах значення всіх параметрів, які характеризують здатність виконувати потрібні функції в заданих режимах та умовах застосування, технічного обслуговування, зберігання та транспортування. Під технічними об'єктами розуміють пристрої, прилади, механізми, машини, комплекси обладнання, будівельні конструкції і споруди, технологічні операції і процеси, системи зв'язку, інформаційні системи, автоматизовані системи управління технологічними процесами тощо [2].

Методи теорії і практики дослідження надійності базуються на застосуванні апарату теорії імовірностей і випадкових процесів, математичної статистики, моделювання.

Теорія надійності [3] — наукова дисципліна, у якій вивчаються і розробляються методи забезпечення ефективності роботи об'єктів (виробів, пристроїв, систем тощо) у процесі експлуатації (рис.1.1). Вона є основою інженерної практики в галузі надійності технічних об'єктів.

В основі теорії надійності і її критеріїв лежить таке поняття як відмова — подія, в результаті якої відбувається повне або часткове порушення роботоздатності.

Під надійністю технічного об'єкта у широкому розумінні слова мається на увазі здатність

технічного пристрою або системи до безвідмовної роботи протягом заданого часу, обумовленого часом виконання поставленого завдання. У справному стані об'єкт повинен відповідати всім вимогам, встановленим для нього нормативно-технічною і конструкторською документацією. Невідповідність хоч би одній з вимог переводить об'єкт в категорію несправних.

В теорії надійності вводяться параметри надійності об'єктів, обґрунтовуються вимоги до надійності з врахуванням економічних та інших факторів, розробляються рекомендації по забезпеченню заданих вимог до надійності на етапах проектування, виробництва зберігання та експлуатації.

Рисунок 1.1 Ефективність роботи об’єкта

В теорії надійності вводяться параметри надійності об'єктів, обґрунтовуються вимоги до надійності з врахуванням економічних та інших факторів, розробляються рекомендації по забезпеченню заданих вимог до надійності на етапах проектування, виробництва зберігання та експлуатації.

Об'єкт критичної інфраструктури — це підприємства та установи (незалежно від форми власності) таких галузей, як енергетика, хімічна промисловість, транспорт, банки та фінанси, інформаційні технології та телекомунікації (електронні комунікації), продовольство, охорона здоров'я, комунальне господарство, що є стратегічно важливими для функціонування економіки і безпеки держави, суспільства та населення, виведення з ладу або руйнування яких може мати вплив на національну безпеку і оборону, природне середовище, призвести до значних матеріальних та фінансових збитків, людських жертв[4].

Програмне забезпечення (програмні засоби) (ПЗ; англ. software) — сукупність програм системи обробки інформації і програмних документів, необхідних для експлуатації цих

програм.

Розрізняють системне програмне забезпечення (зокрема, операційна система, транслятори, редактори, графічний інтерфейс користувача); прикладне програмне забезпечення, що використовується для виконання конкретних завдань, наприклад, статистичне програмне забезпечення; інструментальне програмне забезпечення (комп'ютерні програми, призначені для проектування, розробки, адміністрування і супроводження системного та прикладного програмного забезпечення).

Комплекс програм, які забезпечують управління компонентами комп'ютерної системи, такими як процесор, оперативна пам'ять, пристрої введення-виведення, мережеве обладнання, виступаючи як «міжшаровий інтерфейс», з одного боку якого — апаратура, а з іншого — додатки користувача. На відміну від прикладного програмного забезпечення, системне не вирішує конкретні практичні завдання, а лише забезпечує роботу інших програм, надаючи їм сервісні функції, абстрагуючи деталі апаратної і мікропрограмної реалізації обчислювальної системи, керує апаратними ресурсами обчислювальної системи. Віднесення того чи іншого програмного забезпечення до системного є умовним, і залежить від угод, використовуваних у конкретному контексті. Як правило, до системного програмного забезпечення відносяться операційні системи, широкий клас сполучного програмного забезпечення. [5].

При розробці ПЗ повинні вирішуватися два основні завдання:

– створення надійного (безпомилкового) базового програмного забезпечення;

– проектування і включення програмних засобів для забезпечення живучості при програмних, інформаційних і апаратних відмовах.

Відомим методом підвищення достовірності розроблення ПЗ є застосування об'єктно-орієнтованого і структурного підходів до програмування, перш за все за рахунок зменшення числа проектних помилок. При налагодженні написаних програм важливо ретельно провести їх тестування [6]. Повне тестування програми зазвичай нездійсненно на практиці, а експериментальне тестування програм може бути доказом наявності помилок, але ніколи не доведе їх відсутності. Тому підвищення достовірності обчислювального процесу з урахуванням можливих прихованих помилок в програмі є актуальним завданням.

Суттєвою проблемою практичної реалізації трьохверсійного обчислювального процесу є налагодження і впровадження нових програмних модулів в реальні системи управління.

Найбільш імовірним джерелом систематичних відмов систем управління є дефекти програмних засобів, внесені при розробці, що не виявлені при тестуванні і верифікації і які з’являються при певному наборі вхідних даних або характеристиках фізичних або інформаційних середовищах. Резервування системи з виконанням ідентичних програмних додатків не вирішує проблему усунення таких дефектів, так як вони тиражуються і з’являються одночасно, викликаючи фатальну відмову.

Проблема відмовостійкості в керуючих система розглядалися в таких роботах: R.T. Wood [11] - розглянуто використання диверсійності у ядерній промисловості, В.С. Харченко [16-18] - проаналізовано моделі багатоверсійних інформаційно-управляючих систем, М.А. Ястребенецького [19], В.В. Скляра [20] - переглянуто можливі способи адаптації у багатоверсійних мажоритарно-резервованих комп'ютерних системах управління, А.В Волкова [21], Ю.Н. Соколова [22] - проаналізовано метод оцінки надійності програмно-технічних комплексів, О.В. Брежнєва [23] - розглянуто метод оцінки безпеки критичної енергетичної інфраструктури, А.В. Федухіна [24], Н.В. Якимця [25] - розглянуто моделі проектування відмовостійких цифрових систем та ін.

Незважаючи на значний внесок в розвиток теорії і практики відмовостійкості керуючих систем, проблема підвищення достовірності розроблювального ПЗ вимагає подальших досліджень з урахуванням додаткових можливостей, що надаються новими розробками електронних компонентів і інформаційних технологій.

1.2 Методи забезпечення відмовостійкості

Відмовостійкість або поступова деґрадація (англ. graceful degradation), або поступове скорочення можливостей, або поступовий вихід із роботи, або плавне знижування ефективності це властивість системи (часто комп'ютерної), що дозволяє їй продовжувати правильно діяти у випадку помилки або декількох помилок в деяких її частинах. Якщо при цьому падає якість експлуатації, то це відбувається пропорціонально до серйозності помилки, на відміну від наївно спроектованих систем, в яких навіть маленька помилка спричиняє загальну відмову. Відмовостійкість особливо популярна у високо доступних та життєво критичних системах [7].

В межах окремої системи, відмовостійкість може бути досягнута очікуванням виняткових умов і побудовою системи, що могла б упоратись із ними, і, загалом, метою для самостабілізації є система, що сходиться в напрямку до стану без помилок. Однак, якщо наслідки збоїв системи є катастрофічними, або ціна побудови достатньо надійної системи занадто висока, найкращим рішенням може бути деяка форма дублювання. В будь-якому випадку, якщо наслідки збою катастрофічні, система має бути спроможною використати реверсію, щоб повернутися до безпечного стану. Це подібно до відкочування, але може бути зроблене людиною, якщо вона присутня в процесі.

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.

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.

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.

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.

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.

В даний час в обчислювальних системах широке поширення отримав цілий ряд методів забезпечення відмовостійкості. Незважаючи на їх різноманіття, дані методи можна розділити на три основні групи, які в своїй основі використовують такі види надмірності:

– апаратну надмірність;
– часова надмірність;
– програмна надмірність.

1.2.1 Апаратна надмірність

Апаратну надмірність - (Hardware Redundancy, більш відома як резервування). Існують методи постійного резервування (синтез надлишкових пристроїв, нечутливих до певної кількості помилок) і методи резервування заміщенням (використання системи контролю, яка може діяти безперервно або періодично, в цьому випадку говорять, про так званому функціональному діагностуванні). Виключаючи навіть короткочасний простий, постійне резервування має відносну перевагу в порівнянні з другою групою методів, системи при відмовах;

Резервування у техніці — спосіб забезпечення надійності об'єкта за рахунок використання додаткових засобів та (або) можливостей, надлишкових відносно мінімально необхідних для виконання потрібних функцій [8]. Резервування є універсальним принципом підвищення характеристик безвідмовності функціонування, що знайшов широке застосовування як у природі, так і техніці та технології й інших сторонах людського життя.

Досить часто до технічних систем і засобів ставляться високі вимоги у плані безвідмовності роботи.

Одні пристрої не можна зупиняти через небезпеку, що загрожує людям, які працюють на цих пристроях або з їх використанням. Наприклад, літальні апарати чи системи постачання повітря чи водовідливу у шахтах, зупинка чи вихід з ладу яких може привести до значних жертв.

Інші пристрої недоцільно зупиняти чи допускати самовільну зупинку через значні економічні збитки. Це, наприклад, системи електропостачання або системи забезпечення теплом тощо.

Треті пристрої повинні бути безвідмовними протягом заданого періоду часу з оборонних (військових) міркувань. Сюди відноситься більшість видів військової техніки.

Ці особливості змушують шукати шляхи підвищення надійності та довговічності пристроїв та систем до заданого рівня.

Одним з таких шляхів є резервування елементів, частин систем та (або) систем у цілому. Суть резервування полягає у тому, що до елемента (блоку, системи) приєднуються один або декілька запасних (резервних) елементів (блоків, систем), які по мірі виникнення відмов стають на місце основного і беруть на себе його функцію.

У резервованому виробі відмова настане тоді, коли вийдуть з ладу основний пристрій (елемент) і усі резервні пристрої (елементи). Група елементів системи вважається резервованою, якщо відмова одного або декількох її елементів не порушує нормальної роботи схеми (системи), а решта справних елементів беруть на себе виконання заданої функції. Таке

резервування системи має назву функціональне резервування, коли в разі відмови резервованої системи її функції виконує друга система, яка до цього здійснювала інші функції.

Резервування в технічних системах класифікують за низкою ознак, основні з яких:

– кратність резервування;

– стан резервних елементів до моменту включення їх у роботу;

– рівень резервування;

– можливість сумісної роботи основних і резервних елементів під загальним навантаженням;

– спосіб сполучення основних і резервних елементів.

1.2.2 Часова надмірність

Часова надмірність (Time Redundancy) полягає у використанні певної частини продуктивності комп'ютера для контролю за виконанням програм та відновлення (рестарту) обчислювального процесу (запас часу для повторного виконання операції (наприклад, з подвійним або потрійним прорахунку на обчислювальній машині);

Про часову надмірності кажуть в тих випадках, коли системі в процесі функціонування надається можливість витратити деякий час для відновлення її технічних характеристик. Можна вказати кілька основних джерел резерву часу.

Перш за все він може створюватися за рахунок збільшення часу, який виділяється системі для виконання дорученого їй завдання і званого надалі оперативним або робочим часом.

Другим основним джерелом є запас продуктивності, який дозволяє зменшити мінімальний час виконання завдання і створити резерв без збільшення оперативного часу системи. Запас продуктивності можна утворити, збільшуючи швидкодію елементів системи або об'єднуючи кілька пристроїв низької продуктивності в єдиний комплекс. У системах, результат роботи яких оцінюється обсягом виробленого продукту, резерв часу можна створити за рахунок внутрішніх запасів вихідної продукції. Для систем обробки інформації такою продукцією є оброблена інформація, для систем енергопостачання - електрична енергія, для систем водопостачання- водні ресурси, для автоматичних ліній в машинобудуванні - деталі і вузли і т.д. Для зберігання запасів слід передбачити спеціальні накопичувачі. У зазначених системах ними є пристрої, що запам'ятовують, акумуляторні батареї, резервуари, бункери і т.д. Поки запас не вичерпано, продукція надходить на вихід системи і суміжні з нею системи не помічають часткового і навіть повного припинення її функціонування.

1.2.3 Програмна надмірність

Програмна надмірність - (Software Redundancy) використовується для контролю і забезпечення достовірності найбільш важливих рішень з управління та обробки інформації. Вона полягає в зіставленні результатів обробки однакових вихідних даних різними програмами і виключення спотворення результатів, обумовлених різними аномаліями.

При проектуванні застосовується загальний підхід до побудови компонентів системи. Виходячи з принципу взаємної недовіри, передбачається, що інші компоненти системи і вихідні дані можуть містити помилки, тому першим кроком при проектуванні нової компоненти буде організація перевірки вхідних даних. Проводиться аналіз вхідних даних на допустимість значень. При їх недопустимості програмний компонент завершує своє виконання з формуванням негативного звіту і звітної інформації.

Звітна інформація, як правило, оформляється у вигляді фрази, яка поміщається в звітне поле, що представляє собою виділену в оперативному запам’ятовучому пристрої (RAM) область пам'яті. Принцип заповнення звіту може бути кільцевим або в лінійку, а формування фрази може супроводжуватися індикатором екстреного реагування (фрази по критичних ситуацій і відмов апаратури). Загальна структура фраз:

– ідентифікатор фрази;

– час формування;

– ідентифікатор компоненти, формує фразу;

– номер завдання або функції, при виконанні якої була сформована фраза;

– ідентифікатор типу фрази - за типом помилки або сформований неприпустимою ситуації;

– вміст фрази: робочі параметри, вхідні дані та інша службова інформація, необхідна для подальшого аналізу.

У процесі свого виконання програмні компоненти формують додаткову службову і звітну інформацію, яка оформляється у вигляді спеціальних інформаційних масивів для більш детального аналізу ходу виконання обчислювального процесу і розвитку аномальних ситуацій. Застосування методів введення надмірності в програмне забезпечення підвищує надійність і передбачуваність виконання як окремо компонент системи, так і всієї системи в цілому. Обчислювальна система стає більш стійкою до помилок.

1.2.3.1 Мультиверсійне програмування

Загальновідомо, що в наш час найбільшу загрозу для сучасних обчислювальних систем представляють несправності проектування [10]. Різкий збільшення складності програмного забезпечення привело до того, що стало практично неможливим виявлення і усунення всіх несправностей проектування ПЗ ввести в дію системи в фазу використання. Зокрема, порівняно недавно була виявлена помилка проектування в процесорі Pentium, з яким вже були виготовлені та розповсюджені тисячі комп'ютерів. Ця помилка коштувала фірмі-виробнику $ 475 000 000. Таких прикладів можна навести безліч.

Фундаментальним рішенням проблеми несправностей проектування в ПЗ є принцип різноманітності проектів. Різність проектів виникла як модель дублювання апаратних засобів. Але в апаратурі застосовується просте копіювання каналів обчислення, тому що воно є ефективним засобом маскування випадкових фізичних несправностей ISSN 1028-9763. Інститут проблем математичних машин і системи, 2009, № 4203 апаратних засобів. Тим часом копіювання функціонального блоку (ФБ) ПО буде копіювати і не виявлених в ньому несправності проектування. Тому кожен ФБ ПО, дублює виконання однієї і тієї ж функції, повинен бути розроблений автономно від інших блоків.

Мультіверсійне програмування, починаючи з 1977 року, використовується там, де потрібна видача точного результату в реальному часі. Причому, на відміну від інших методик програмної відмовостійкості, вимога до часу видачі результату є більш жорстким. Очевидно, що при створенні систем з використанням даної концепції будуть виникати труднощі, в першу чергу, пов'язані з побудовою середовища, яка буде виробляти мультіверсійного виконання програмних модулів.

Основою побудови мультіверсійного середовища виконання є опрацювання її структури і зв'язків між елементами, так як від цього залежить не тільки здатність розширюваності і зручність у використанні середовища, але і такі параметри як надійність і продуктивність.

Існує кілька методів мінімізації помилок в програмному забезпеченні. Одним з них є мультиверсійне програмування [9], що являє собою незалежну генерацію $N \geq 3$ функціонально еквівалентних програм (мультіверсій) відповідно з ідентичними вихідними специфікаціями на проектування. Для цих N програм надані засоби конкурентного виконання, по ходу якого в певних точках контролю ("cc-points" от cross-check points, точки перехресного контролю) програмами генеруються вектори порівняння ("c-vectors" від comparison vectors, вектори порівняння). Складові векторів порівняння і контрольні точки генерації "с-векторів" попередньо визначені ще на етапі вихідних специфікацій. Мультиверсійне програмування дозволяє знизити ризики систематичних відмов, оскільки в разі проектних дефектів зменшується ймовірність одночасного і однотипного відмови різних версій ПЗ.

В цьому і полягає поняття різноманітності проектування програмного забезпечення або багатоверсійних обчислень. ФБ ПЗ, не забезпечений засобами відмовостійкості, називається симплексним ФБ [10]. Він навіть після тестування може мати не виявлені недіючі несправності, при певних обставинах (як наприклад, при деякому випадковому наборі вхідних даних) активізуються і призводять до вироблення ФБ неправильних вихідних результатів. У цьому випадку ми будемо мати дефектний ФБ ПЗ. Щоб зробити такий ФБ відмовостійким, до нього додають один або кілька симплексних ФБ, які будуть маскувати результат несправного ФБ. Для того, щоб забезпечити відмовостійкість набору з N ФБ, де N = 3, потрібно побудувати для нього середовище відмовостійкого виконання (рис. 1.2).

Це середовище має забезпечувати порівняння результатів роботи ФБ в навмисне обраних точках перехресного контролю.

Для оцінки ефективності запропонованого способу збільшення відмовостійкості ПО можна використовувати методику, наведену в [11].

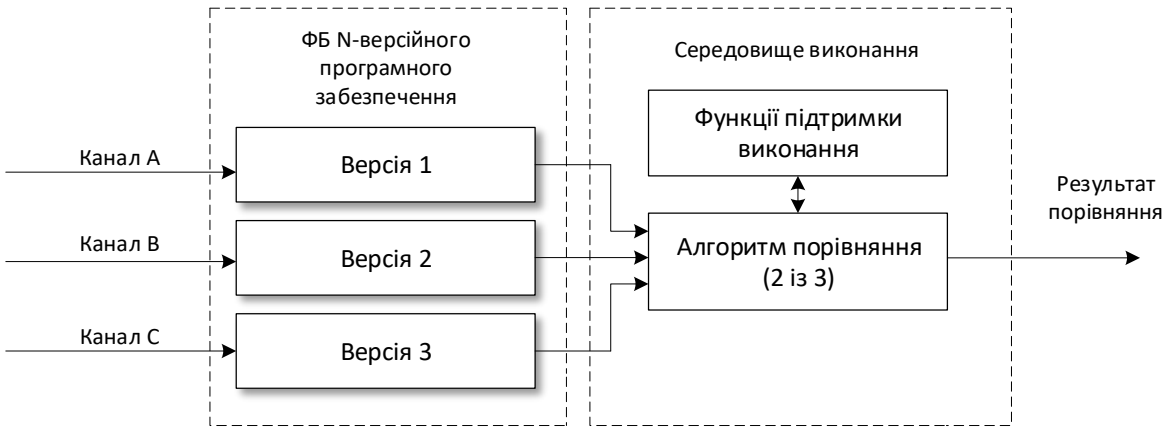


Рисунок 1.2 Модель N -версійного програмного забезпечення

Для роботи ПЗ в режимі N - версійного виконання потрібен також механізм синхронізації. За допомогою цього механізму кожна версія сповіщає про готовність свого вектора і переходить в режим очікування сигналу від механізму вибору рішення. Механізм синхронізації також запобігає голосуванню результатів доти, поки не будуть готові результати (вектори) всіх версій. Дії, які повинні бути виконані у відповідь на результати вирішення в точках перехресного контролю, залежать від того, всі версії виробили відповідає вимогам результат (вектор) в межах заданого часу, і від того, чи збігаються ці результати. Такими діями можуть бути: продовження виконання версії, завершення виконання версії, продовження після заміни вектора дефектної версії на вектор, прийнятий в результаті рішення.

1.3 Використання мультиядерного мікропроцесору для реалізації трьохверсійного обчислювального процесу

В даний час в системах управління широкого поширення набули контролери з мікропроцесорами фірми Intel.

Поява високошвидкісних багатоядерних мікропроцесорів (англ. Multi-core) створило передумови для практичної реалізації мультіверсійного обчислювального процесу за рахунок апаратної та тимчасової надмірності.

Багатоядерний процесор складається з двох і більше «обчислювальних ядер» на одному кристалі. Він має один корпус і встановлюється в один роз'єм на системній платі комп'ютера, але операційна система сприймає кожне його обчислювальне ядро як окремий процесор з повним набором обчислювальних ресурсів.

У всіх існуючих на сьогоднішній день багатоядерних процесорах кеш-пам'ять першого рівня у кожного ядра своя, а кеш 2-го рівня існує в кількох варіантах:

– поділюваних - кеш розташований на одному з обома ядрами кристалі і доступний кожному з них у повному обсязі. Використовується в процесорах сімейств Intel Core.

– індивідуальний - окремі кеші рівного обсягу, інтегровані в кожне з ядер. Обмін даними з кешей L2 між ядрами здійснюється через контролер пам'яті — інтегрований (Athlon 64 X2) або зовнішній (Pentium D).

У застосунках, оптимізованих під паралельне виконання, спостерігається приріст продуктивності на двоядерних процесорах. Однак, якщо програма не оптимізована, то воно не буде отримувати практично ніякої вигоди від додаткових ядер, а може навіть виконуватися повільніше, ніж на процесорі з меншою кількістю ядер, але більшою тактовою частотою. Це в основному старі програми, або програми, яким багатозадачність не потрібна (наприклад, програвач музики) або неможлива.

На сьогоднішній день основні виробники процесорів — Intel і AMD — визнали подальше збільшення числа ядер процесорів як один з пріоритетних напрямів збільшення продуктивності. Компанією AMD вже освоєно виробництво 8-ядерних процесорів для домашніх комп'ютерів, а також 16-ядерних в серверних системах. Intel у цьому показнику дещо відстає - освоєно виробництво 6- та 10-ядерних відповідно, але кожен з них отримує по 2 потоки команд, тобто віртуально їх вдвічі більше (Hyper-Threading) [27], що дає 10-15% приросту швидкодії.

Тому було розглянуто реалізацію надійного обчислювального процесу на прикладі програмного забезпечення, що виконується в контролері з чотирьохядерним процесором по схемі «2 із 3». Вибір схеми обробки по схемі «2 із 3» обумовлений наступним факторами: схеми типу «1 із N» (N=1,2,…) не відповідають вимогам відповідних стандартів (наприклад, [12]) і не

можуть застосовуватися в системах управління реального часу (не володіють функціональною безпекою).

Схеми типу «N із N» (N = 1,2, ...) не мають великої надійності. Наприклад, користуючись [13], Можна розрахувати, що в схемі «2 із 2» при ймовірності появи прихованого дефекту в циклі управління $p=10^{-7}$ в одній з двох гілок програми середній час напрацювання на відмову системи управління рівне Tср≈70 h при часу циклу t = 0,05 s.

Схема «2 із 3» при тій же ймовірності появи прихованого дефекту забезпечує середній час напрацювання на відмову Tср≈2640 років. Застосування схеми «2 із 3» (на відміну від схеми «2 із 2») дозволяє також однозначно визначати гілку програми з проявленим дефектом і міняти результат виконання цієї гілки на вірний. Застосування схем «2 із N» (N = 4,5, ...) недоцільно, тому що на практиці не потрібно такі наднадійні системи і вони складні в реалізації.

1.4 Програмне забезпечення систем управління

Програмне забезпечення систем управління, що функціонує в контролерах, складається з двох блоків - системного і прикладного програмного забезпечення. Узагальнена структурна схема програмного забезпечення представлена на рис. 1.3.

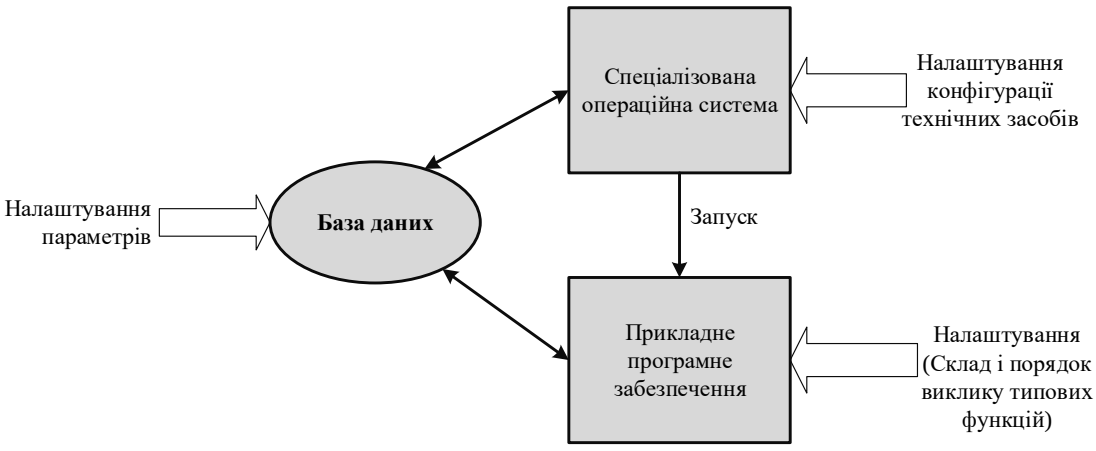


Рисунок 1.3 – Узагальнена структурна схема програмного забезпечення

Ядром системного ПЗ є спеціалізована операційна система (ОС) реального часу.

До складу операційної системи входять:

– ядро операційної системи, що забезпечує розподіл та управління ресурсами обчислювальної системи;

– базовий набір прикладних програм, системні бібліотеки та програми обслуговування.

Ядро системи — це набір функцій, структур даних та окремих програмних модулів, які завантажуються в пам'ять комп'ютера при завантаженні операційної системи та забезпечують три типи системних сервісів:

– управління введенням-виведенням інформації (підсистема вводу-виводу ядра ОС);

– управління оперативною пам'яттю (підсистема управління оперативною пам'яттю ядра ОС);

– управління процесами (підсистема управління процесами ядра ОС).

Кожна з цих підсистем представлена відповідними функціями ядра системи.

Багатозадачні операційні системи також включають ще одну обов'язкову складову — механізм підтримки багатозадачності. Ця складова не надається як системний сервіс і тому не може бути віднесена до жодної з підсистем.

Існує три основних механізми забезпечення багатозадачності (планування задач):

– шляхом надання процесора окремій задачі на квант часу, який визначається самою задачею (кооперативна багатозадачність; останнім часом практично не використовується або область використання значно обмежена всередині процесів);

– шляхом надання процесора окремій задачі на квант часу, який визначається обладнанням обчислювальної системи — інтервальним таймером;

– виділення під окрему задачу окремого процесора в багатопроцесорних системах.

У перших двох випадках на кожному з процесорів в окремо взятий момент часу обраховується лише одна задача, але за рахунок достатньо малого кванту часу (в межах мілісекунд), що почергово надається кожній з задач, виникає ілюзія одночасного виконання в системі багатьох задач.

В сучасних системах, як правило комбінується останні два методи. [14]

Операційна система визначає:

– підтримку функціонування прикладного програмного забезпечення відповідно до налаштованих модулів операційної системи на конфігурацію конкретного виконання і необхідний набір функцій;

– обмін інформацією з зовнішніми абонентами по лініях зв'язку мережевих контролерів;

– обмін інформацією з модулями зв'язку з об'єктами за допомогою введення даних і формування керуючих впливів на об'єкт;

– контроль і захист від несанкціонованого доступу;

– забезпечує можливість паралельного в часі функціонування в рамках одного мікроконтролера (на трьох ядрах) трьох ідентичних по функції гілок однієї програми з процедурами порівняння результатів виконання цих гілок і прийняттям рішень щодо подальшого функціонування за результатами порівняння. Даний режим функціонування забезпечує диверсійність реалізації прикладного програмного забезпечення; контроль працездатності технічних і програмних засобів в процесі функціонування;

– перехід в безпечний режим функціонування при відмові.

Прикладне ПЗ реалізує функції системи управління. Воно складається з моделі і пакетів прикладних програм, що реалізують типові функції:

– управління об’єктом;

– перевірка умов безпеки;

– контроль над об’єктом.

Пакети прикладних програм можна умовно розділити на кілька типових програм, що вирішують основні функції та функції взаємодії з персоналом і обладнанням.

База даних включає наступні групи параметрів:

– дані про поточний стан об'єктів;

– команди управління від користувача;

– параметри функціонування прикладних програм, сформовані на попередньому такті роботи;

– узагальнені дані про поточний стан об'єктів і результати їх контролю;

– дані результатів перевірки умов безпеки;

– команди управління для об'єктів;

– параметри функціонування прикладних програм, сформовані для наступного такту роботи.

Аналіз варіантів побудови програмного забезпечення показує, що для забезпечення захисту систем управління від систематичних відмов доцільно застосування варіанту, який заснований на принципі «Різні алгоритми, логіка и програмна структура» із залученням трьох незалежних між собою програмістів. При виборі мов с-версійного програмування необхідно керуватися стандартом IEC 61131-3 [15], загальновизнаного в світі стандарту по мовам технологічного програмування для програмованих контролерів.

1.5 Постановка завдання

Найбільш імовірним джерелом систематичних відмов систем управління є приховані дефекти програмних засобів, внесені при розробці, що не виявлені при тестуванні і верифікації і які з’являються при певному наборі вхідних даних або характеристиках фізичних або інформаційних середовищах. Резервування системи з виконанням ідентичних програмних додатків не вирішує проблему усунення таких дефектів, так як вони тиражуються і з’являються одночасно, викликаючи фатальну відмову.

Однією з основних проблем при розробці систем управління критичними об'єктами є створення високонадійного ПЗ керуючих контролерів. При розробці ПЗ повинні вирішуватися два основні завдання:

– створення надійного (безпомилкового) базового програмного забезпечення;

– проектування і включення програмних засобів для забезпечення живучості при програмних, інформаційних і апаратних відмовах.

Відомим методом підвищення достовірності розроблювального ПЗ є застосування об'єктно-орієнтованого і структурного підходів до програмування, перш за все за рахунок зменшення числа проектних помилок. При налагодженні написаних програм важливо ретельно провести їх тестування. Повне тестування програми зазвичай нездійсненно на практиці, а експериментальне тестування програм може бути доказом наявності помилок, але ніколи не доведе їх відсутності. Тому підвищення достовірності обчислювального процесу з урахуванням можливих прихованих помилок в програмі є актуальним завданням.

В даний час в системах управління широкого поширення набули контролери з мікропроцесорами фірми Intel. Поява високошвидкісних багатоядерних мікропроцесорів (англ. Multi-core) створило передумови для практичної реалізації мультиверсійного обчислювального процесу за рахунок апаратної та тимчасової надмірності.

1.6.1 Технічне завдання

Метою даної роботи є дослідження способів організації надійного відповостійкого обчислювального процесу в керуючому промисловому контролері і практична реалізація мультіверсійного програмування.

Основні задачі магістерської роботи:

– аналіз існуючих методів забезпечення відмовостійкості;

– побудова моделі забезпечення відмовостійкості шляхом впровадження мультиверсійності;

– реалізація програмного забезпечення – диспетчера відмовостійкої керуючої системи.

Об'єкт дослідження – програмне забезпечення промислового контролера.

Предмет дослідження – методи реалізації трьохверсійності в мультиядерному контролері для підвищення відмовостійкості керуючих систем.

Наукова новизна. Удосконалено модель диспетчера керуючого обчислювального процесу шляхом дослідження організації паралельних обчислювальних процесів у мультиядерних мікропроцесорах, що дозволяє підвищити загальну відмовостійкість і зменшити витрати при впровадження її у систему.

1.7 Висновки по першого розділу

У першому розділі розглянуто проблема створення високонадійного програмного забезпечення керуючих контролерів. При розробці високонадійного ПЗ повинні вирішуватися два основні завдання:

– створення надійного (безпомилкового) базового програмного забезпечення;

– проектування і включення програмних засобів для забезпечення живучості при програмних, інформаційних і апаратних відмовах.

Проаналізовано існуючі методи забезпечення відмовостійкості та сформульована постановка задачі.

2 АНАЛІЗ РЕАЛІЗАЦІЇ ДИСПЕТЧЕРА ОПЕРАЦІЙНОЇ СИСТЕМИ РЕАЛЬНОГО ЧАСУ

Даний розділ присвячується дослідженню операційної системи реального часу (ОСРЧ) та її диспетчера, оцінці обранного мікропроцесору та специфікації MPI. В розділі розглядаються три архітектури ОСРЧ, функції диспетчера, будова мікроконтролера та програмні методи забезпечення паралельного програмування. На підставі аналізу виявлено переваги обраної архітектури ОСРЧ, переваги обраного мікроконтролера та задовільність використанної специфікації MPI.

2.1 Операційна система реального часу

Операційна система реального часу (ОСРЧ, англ. Real-time operating system, RTOS) — тип операційної системи, основне призначення якої — надання необхідного та достатнього набору функцій для роботи систем реального часу на конкретному апаратному обладнанні.

Специфікація UNIX в редакції 2 дає наступне визначення:

Реальний час в операційних системах — це здатність операційної системи забезпечити необхідний рівень сервісу в певний проміжок часу [26].

Ідеальна ОСРЧ має передбачувану поведінку при всіх сценаріях навантаження, включаючи одночасні переривання і виконання потоків [27].

2.1.1 Системи жорсткого та м'якого реального часу

Операційні системи реального часу іноді ділять на два типи — системи жорсткого реального часу та системи м'якого реального часу [28].

Операційна система, яка може забезпечити необхідний час виконання завдання реального часу навіть в найгірших випадках, називається операційною системою жорсткого реального часу. Система, яка може забезпечити необхідний час виконання завдання реального часу в середньому, називається операційною системою м'якого реального часу.

Системи жорсткого реального часу не дозволяють затримок реакції системи, так як це може призвести до втрати актуальності результатів, великих фінансових втрат або навіть аварій і катастроф (рис.2.1). Ситуація, в якій обробка подій відбувається за час, більший передбаченого, в системі жорсткого реального часу вважається фатальною помилкою. При виникненні такої ситуації операційна система перериває операцію і блокує її, щоб, наскільки

можливо, не постраждала надійність і готовність іншої частини системи. Прикладами систем жорсткого реального часу можуть бути бортові системи управління (на літаку, космічному апараті, кораблі, та ін.), системи аварійного захисту, реєстратори аварійних подій [29].

В системі м'якого реального часу затримка реакції вважається відновлювальною помилкою, яка може привести до збільшення вартості результатів і зниження продуктивності, але не є фатальною. Прикладом може служити робота комп'ютерної мережі [30]. Якщо система не встигла обробити черговий прийнятий пакет, це призведе до зупинки на стороні передачі та повторної відправки даних (в залежності від протоколу передачі даних). Дані при цьому не втрачаються, але продуктивність мережі знижується.

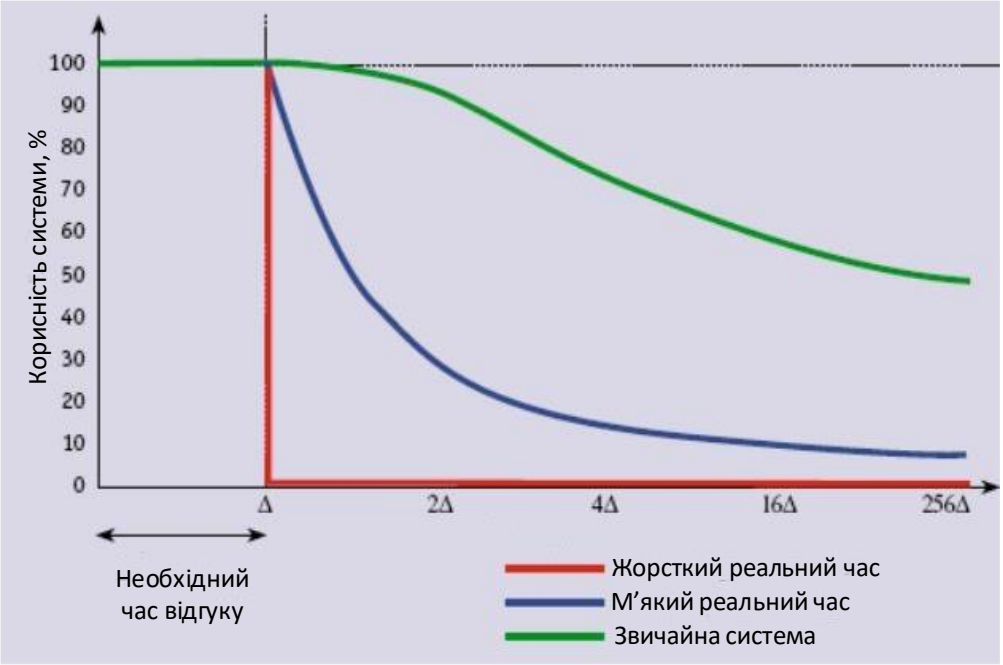


Рисунок 2.1 Відмінність жорстої, м'якої та звичайної системи

Основна відмінність систем жорсткого і м'якого реального часу можна охарактеризувати так: система жорсткого реального часу будь-коли запізниться з реакцією на подію, система м'якого реального часу не повинна відставати від реакції на подію [31].

Часто операційною системою реального часу вважають лише систему, яка може бути використана для вирішення завдань жорсткого реального часу. Це визначення означає наявність у ОСРЧ необхідних інструментів, але також означає, що ці інструменти необхідно правильно використовувати [32].

Більшість програмного забезпечення орієнтоване на «м'який» реальний час. Для подібних систем характерно:

– гарантований час реакції на зовнішні події (переривання від устаткування);

– жорстка підсистема планування процесів (високопріоритетні завдання не повинні витіснятися фонової, за деякими винятками);

– підвищені вимоги до часу реакції на зовнішні події або реактивності (затримка виклику обробника переривання не більше десятків мікросекунд, затримка при перемиканні задач не більше сотень мікросекунд).

Класичним прикладом завдання, де потрібно ОСРЧ, є управління роботом, що беруть деталь зі стрічки конвеєра. Деталь рухається, і робот має лише маленький проміжок часу, коли він може її взяти. Якщо він запізниться, то деталь вже не буде на потрібній ділянці конвеєра, і отже, робота не буде виконана, незважаючи на те, що робот знаходиться в правильному місці. Якщо він підготується раніше, то деталь ще не встигне під'їхати, і він заблокує їй шлях.

Також для операційних систем іноді використовується поняття «інтерактивного реального часу», в якому визначається мінімальний поріг реакції на події графічного інтерфейсу, протягом якого оператор — людина — здатний спокійно, без нервозності, очікувати реакції системи на дані їм вказівки.

2.1.2 Відмінні риси ОСРЧ

Таблиця 2.1 порівняння ОСРЧ і звичайних операційних систем [33].

	ОС реального часу	ОС загального призначення
Основна задача	Встигнути зреагувати на події, що відбуваються на устаткуванні	Оптимально розподілити ресурси комп'ютера між користувачами та задачами
На що орієнтована	Обробка зовнішнії подій	Обробка дій користувача
Як позиціонується	Інструмент для створення конкретного апаратно-програмного комплексу реального часу	Сприймається користувачем як набір додатків, готових до використання
Кому призначена	Кваліфікований розробник	Користувач середньої кваліфікації

2.1.3 Архітектури ОСРЧ

У своєму розвитку ОСРЧ будувалися на основі наступних архітектур:

– монолітна архітектура (рис. 2.2). ОС визначається як набір модулів, взаємодіючих між собою всередині ядра системи і надають прикладному ПО вхідні інтерфейси для звернень до апаратури. Основний недолік цього принципу побудови ОС полягає в поганій передбаченості її поведінки, викликаної складною взаємодією модулів між собою;

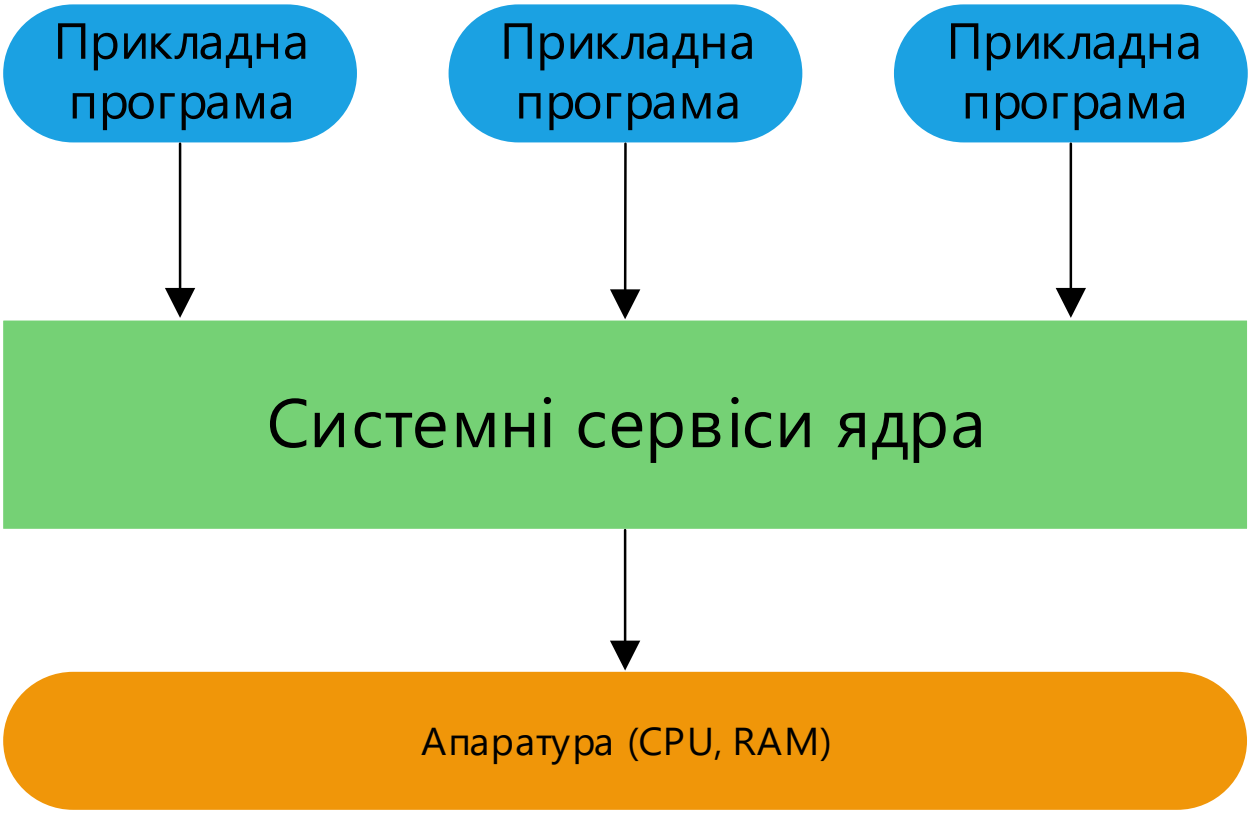


Рисунок 2.2 Монолітна архітектура

– рівнева (шарова) архітектура (рис 2.3). Прикладне ПО має можливість отримати доступ до апаратури не тільки через ядро системи та її сервіси, а й безпосередньо. У порівнянні з монолітною така архітектура забезпечує значно більший ступінь передбачуваності реакцій системи, а також дозволяє здійснювати швидкий доступ прикладних програм до апаратури. Головним недоліком таких систем є відсутність багатозадачності;

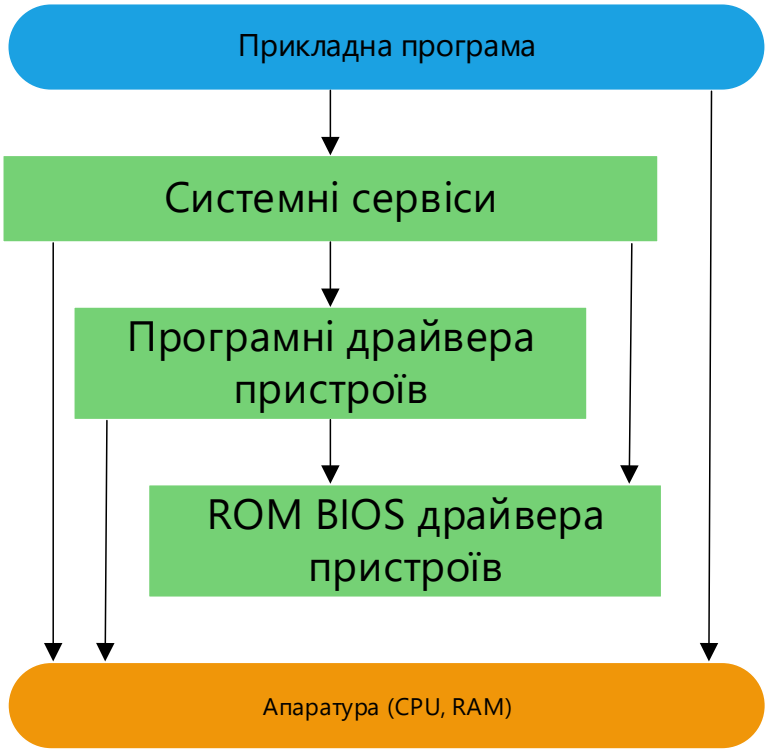


Рисунок 2.3 Рівнева архітектура

– архітектура «клієнт-сервер» (рис.2.4). Основний її принцип полягає у винесенні сервісів ОС у вигляді серверів на рівень користувача та виконанні мікроядром функцій диспетчера повідомлень між клієнтськими користувача програмами і серверами — системними сервісами.

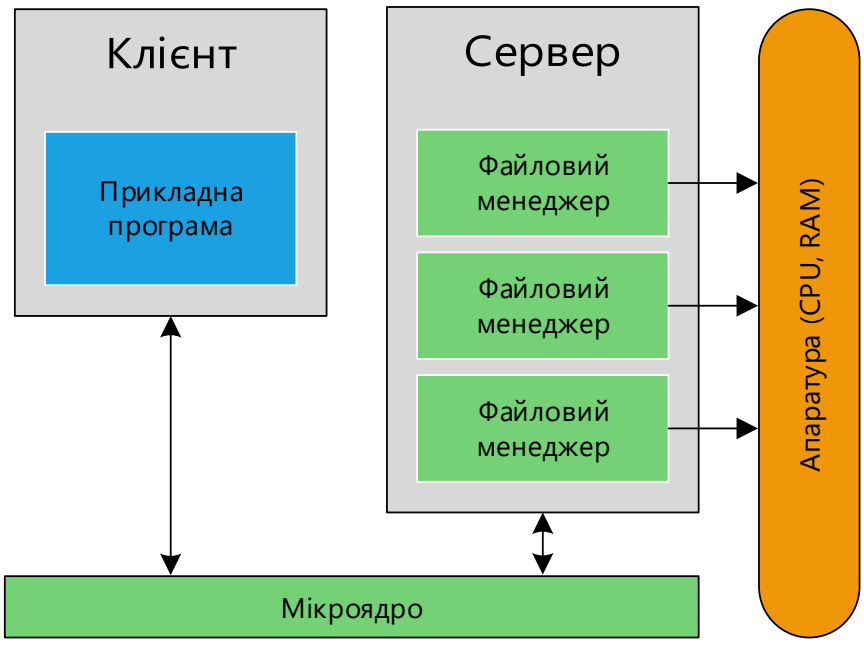


Рисунок 2.4 Архітектура «клієнт-сервер»

Переваги такої архітектури:

1. підвищена надійність, так як кожен сервіс є, по суті, самостійним додатком і його потрібно легше налагодити і відстежити помилки;
2. покращена масштабованість, оскільки непотрібні сервіси можуть бути виключені з системи без шкоди до її працездатності;
3. підвищена відмовостійкість, так як сервіс, який завис може бути перезапущений без перезавантаження системи.

2.1.4 Особливості ядра

Ядро ОСРЧ Забезпечує функціонування проміжного абстрактного рівня ОС, який приховує від прикладного ПО специфіку технічного пристрою процесора (декількох процесорів) і пов'язаного з ним апаратного забезпечення [34].

Основні сервіси

Зазначений абстрактний рівень надає для прикладного програмного забезпечення п'ять основних категорій сервісів [34] [35]:

– управління задачами. Найголовніша група сервісів. Дозволяє розробникам додатків проектувати програмні продукти у вигляді наборів окремих програмних фрагментів, кожен з яких може ставитися до своєї тематичної області, виконувати окрему функцію і мати свій власний квант часу, відведений йому для роботи. Кожен такий фрагмент називається задачою. Сервіси в розглянутій групі мають здатність запускати задачі і присвоювати їм пріоритети. Основний сервіс тут — планувальник задач. Він здійснює контроль над виконанням поточних завдань, запускає нові в відповідний період часу і стежить за режимом їх роботи;

– динамічний розподіл пам'яті. Багато (але не всі) ядра ОСРЧ підтримують цю групу сервісів. Вона дозволяє завданням запозичувати області оперативної пам'яті для тимчасового використання в роботі додатків. Часто ці області згодом переходять від завдання до завдання, і за допомогою цього здійснюється швидка передача великої кількості даних між ними. Деякі дуже малі за розміром ядра ОСРЧ, які передбачається використовувати в апаратних середовищах із суворим обмеженням на обсяг використовуваної пам'яті, не підтримують сервіси динамічного розподілу пам'яті;

– управління таймерами. Так як вбудовані системи пред'являють жорсткі вимоги до тимчасових рамок виконання завдань, до складу ядра ОСРЧ включається група сервісів, які забезпечують управління таймерами для відстеження ліміту часу, протягом якого повинна виконуватися завдання. Ці сервіси вимірюють і задають різні проміжки часу (від 1 мкс і вище),

генерують переривання після закінчення тимчасових інтервалів і створюють разові і циклічні будильники;

– взаємодія між задачами та синхронізація. Сервіси даної групи дозволяють завданням обмінюватися інформацією та забезпечують її збереження. Вони також дають можливість програмним фрагментами узгоджувати між собою свою роботу для підвищення ефективності. Якщо виключити ці сервіси зі складу ядра ОСРЧ, то задачі почнуть обмінюватися спотвореною інформацією і можуть стати перешкодою для роботи сусідніх задач;

– контроль пристрою введення-виведення. Сервіси цієї групи забезпечують роботу єдиного програмного інтерфейсу, взаємодіє з усім безліччю драйверів пристроїв, які є типовими для більшості вбудованих систем.

На додаток до сервісів ядра, багато ОСРЧ пропонують лінійки додаткових компонентів для організації таких високорівневих понять, як файлова система, мережева взаємодія, управління мережею, управління базою даних, графічний, призначений для користувача, інтерфейс і т. д. Хоча багато з цих компонентів набагато більше і складніше, ніж саме ядро ОСРЧ, вони, тим не менш, ґрунтуються на його сервісах. Кожен з таких компонентів включається через вбудовану систему, тільки якщо її сервіси необхідні для виконання вбудованого додатка і тільки для того, щоб звести витрати пам'яті до мінімуму [34].

2.1.5 Відмінності ОСРЧ від операційних систем загального призначення

Багато операційних систем загального призначення також підтримують зазначені вище сервіси. Однак ключовою відмінністю сервісів ядра ОСРЧ є детермінований, заснований на суворому контролі часу, характер їх роботи. В даному випадку під детермінованістю розуміється те, що для виконання одного сервісу операційної системи потрібно часовий інтервал свідомо відомої тривалості. Теоретично цей час може бути обчислено за математичними формулами, які повинні бути жорстко алгебраїчними і не повинні включати ніяких часових параметрів випадкового характеру. Будь-яка випадкова величина, яка визначає час виконання завдання в ОСРЧ, може викликати небажану затримку в роботі додатка, тоді наступна задача не вкладеться в свій квант часу, що послужить причиною для помилки[34].

2.1.6 Планування задач

2.1.6.1 Работа диспетчера

Більшість ОСРЧ виконує планування задач, керуючись наступною схемою [34]. Кожній задачі в додатку ставиться у відповідність деякий пріоритет. Чим більше пріоритет, тим вище повинна бути реактивність завдання. Висока реактивність досягається шляхом реалізації підходу пріоритетного витісняющого планування (preemptive priority scheduling), суть якого полягає в тому, що за розкладом дозволяється зупиняти виконання будь-якої задачі в довільний момент часу, якщо встановлено, що інша задача повинна бути запущена негайно. Описана схема працює за наступним правилом: якщо дві задачі одночасно готові до запуску, але перша володіє високим пріоритетом, а друга — низьким, то планувальник віддасть перевагу першій. Друга задача буде запущена тільки після того, як завершить свою роботу перша.

Можлива ситуація, коли задача з низьким пріоритетом вже запущена, а планувальник отримує повідомлення, що інша задача з більш високим пріоритетом готова до запуску. Причиною цього може послужити будь-яка зовнішня дія (переривання від устаткування), як, наприклад, зміна стану перемикача пристрою під керуванням ОСРЧ. У такій ситуації планувальник задач діє згідно підходу пріоритетного витісняющого планування наступним чином. Задачі з низьким пріоритетом буде дозволено виконати до кінця поточну машинну команду (але не команду, описану в програмі мовою високого рівня), після чого виконання завдання призупиняється [34]. Далі запускається задача з високим пріоритетом. Після того, як вона опрацьовує, планувальник запускає перервану першу задачу з машинної команди, наступної за останньою виконаною.

Кожен раз, коли планувальник задач отримує сигнал про настання деякого зовнішньої події (тригер), причина якого може бути як апаратна, так і програмна, він діє за наступним алгоритмом [34]:

1. Визначає, чи повинна поточна виконувана задача продовжувати працювати.
2. Встановлює, яка задача має запускатися наступною.
3. Зберігає контекст зупиненої задачі (щоб вона потім відновила роботу з місця зупинки).
4. Встановлює контекст для наступного завдання.
5. Запускає цю задачу.

Ці п'ять кроків алгоритму також називаються перемиканням задач.

2.1.6.2 Виконання задачі

У звичайних ОСРЧ задача може перебувати в трьох можливих станах:

– задача виконується;

– задача готова до виконання;

– задача заблокована.

Велику частину часу основна маса завдань заблокована. Тільки одне завдання може виконуватися на центральному процесорі в поточний момент часу. У примітивних ОСРЧ список готових до виконання завдань, як правило, дуже короткий, він може складатися не більше ніж з двох-трьох найменувань.

Основна функція адміністратора ОСРЧ полягає в організації такого планувальника задач.

Якщо в списку готових до виконання задач є більше двох—трьох останніх, то передбачається, що всі задачі розташовані в оптимальному порядку. Якщо ж трапляються такі ситуації, що число завдань в списку перевищує допустимий ліміт, то задачі сортуються в порядку пріоритету.

2.1.6.3 Алгоритми планування

В даний час для вирішення завдання ефективного планування в ОСРЧ найбільш інтенсивно розвиваються два підходи [36]:

– статичні алгоритми планування (RMS, Rate Monotonic Scheduling) — використовують пріоритетне витісняють планування, пріоритет надається кожній задачі до того, як вона почала виконуватися, перевага віддається задачам з найкоротшими періодами виконання;

– динамічні алгоритми планування (EDF, Earliest Deadline First Scheduling) — пріоритет задачам привласнюється динамічно, причому перевага віддається задачам з найбільш раннім граничним часом початку (завершення) виконання.

При великих навантаженнях системи EDF більш ефективний, ніж RMS.

2.1.7 Взаємодія між завданнями та розподіл ресурсів

Багатозадачним системам необхідно розподіляти доступ до ресурсів. Одночасний доступ двох і більше процесів до будь-якої області пам'яті або інших ресурсів являє певну загрозу.

Існує три способи вирішення цієї проблеми: тимчасове блокування переривань, виконавчі семафори, передача сигналів. ОСРЧ зазвичай не використовують перший спосіб, тому що додаток користувача не може контролювати процесор стільки, скільки хоче. Однак у багатьох вбудованих системах і ОСРЧ дозволяється запускати додатки в режимі ядра для доступу до системних викликів і дається контроль над оточенням виконання без втручання ОС.

На однопроцесорних системах найкращим рішенням є додаток, запущений в режимі ядра, якому дозволено блокування переривань. Поки переривання заблоковано, додаток використовує ресурси процесу одноосібно і ніяка інша задача або переривання не може виконуватися. Таким чином захищаються всі критичні ресурси. Після того як додаток завершить критичні дії, він повинен розблокувати переривання, якщо такі є. Тимчасове блокування переривання дозволено тільки тоді, коли найдовший проміжок виконання критичної секції менше, ніж допустимий час реакції на переривання. Зазвичай цей метод захисту використовується, тільки коли довжина критичного коду не перевищує декількох рядків і не містить циклів. Цей метод ідеально підходить для захисту регістрів.

Коли довжина критичної ділянки більше максимальної або містить цикли, програміст повинен використовувати механізми, ідентичні або імітуючи поведінку систем загального призначення, такі, як семафори і передача сигналів.

2.1.8 Виділення пам'яті

Наступним проблемам виділення пам'яті в ОСРЧ приділяється більше уваги, ніж в операційних системах загального призначення.

По-перше, швидкості виділення пам'яті. Стандартна схема виділення пам'яті передбачає сканування списку невизначеної довжини для знаходження вільної області пам'яті заданого розміру, а це неприйнятно, тому що в ОСРЧ виділення пам'яті повинно відбуватися за фіксований час.

По-друге, пам'ять може стати фрагментованою в разі розподілу вільних її ділянок вже запущеними процесами. Це може привести до зупинки програми через її нездатність задіяти нову ділянку пам'яті. Алгоритм виділення пам'яті, поступово збільшує фрагментованість пам'яті, може успішно працювати на настільних системах, якщо ті перезавантажуються не рідше одного разу на місяць, але є неприйнятним для вбудованих систем, які працюють роками без перезавантаження.

Простий алгоритм з фіксованою довжиною ділянок пам'яті дуже добре працює в нескладних вбудованих системах.

Також цей алгоритм відмінно функціонує і в настільних системах, особливо тоді, коли

під час обробки ділянки пам'яті одним ядром наступна ділянка пам'яті обробляється іншим ядром. Такі оптимізовані для настільних систем ОСРЧ, як Unison Operating System або DSPnano RTOS, можуть надавати таку можливість.

2.2 Архітектура обранного микропроцесора

Паралельне виконання мультиверсій диспетчера у ОСРЧ на пряму залежить від архітектури обранного мікропроцесору. Ключовими факторами при виборі мікропроцесору є:

– кількість ядер процесору та їх частота;
– наявність кешу у кожного із ядер та їх розмір;
– архітектура роботи із загальною оперативною пам’яттю.

Тому необхідно ретельно вивчити надану архітектуру, щоб знати її плюси та мінуси.

Для дослідження методів відмовостійкості, котре полягає у реализації мультиверсійного ПЗ, було обрано процесор Intel Core i5-7300HQ - 4-ядерний процесор з тактовою частотою 2500 MHz і кешем 3-го рівня 6144 KB. Процесор призначений для мобільних комп'ютерів, роз'єм - BGA1440. Має вбудований контролер оперативну пам'ять (2 канали, DDR3L-1600, LPDDR3-2133, DDR4-2400) і контролер PCI Express 3.0 (кількість ліній - 16).

2.2.1 Архітектура з неоднорідним доступом до пам'яті (NUMA)

NUMA (Non-Uniform Memory Access, архітектура з неоднорідним доступом до пам'яті) — організація пам'яті в паралельних архітектурах ЕОМ, в якій над фізично розподіленою між окремими обчислювальними блоками пам'яттю створюється спільний адресний простір (найчастіше програмним способом) таким чином, що на рівні програмної організації (архітектури) пам'ять системи сприймається як загальна для усіх процесорів [37]. Однак доступ до різних її областей займає різний час (тому, власне, ці архітектури називаються архітектурами з неоднорідним доступом до пам'яті) (рис. 2.6).

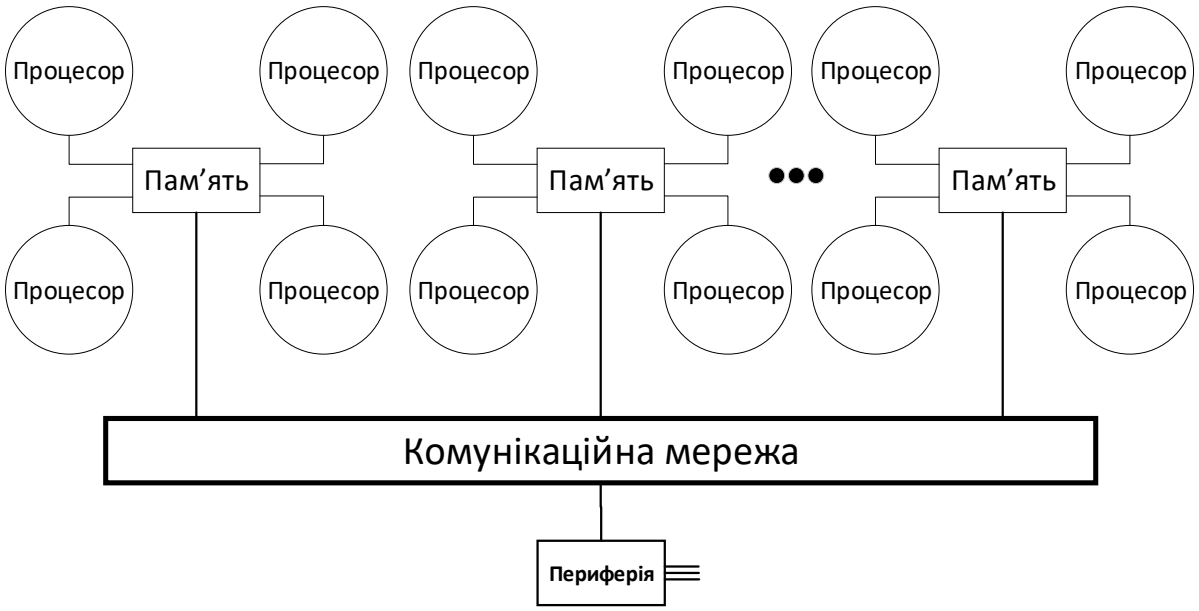


Рисунок 2.6 Архітектура з неоднорідним доступом до пам'яті (NUMA)

Значною проблемою таких архітектурних рішень є забезпечення цілісності даних та своєчасної синхронізації змін в даних, які стали результатом якихось локальних обчислень, на всю глобальну пам'ять системи. Ця проблема знана під назвою проблеми когерентності кеш-пам'яті. Архітектури з когерентною кеш-пам'яттю носять абревіатуру ccNUMA (cache-coherent NUMA).

NUMA є певним компромісом між паралельними архітектурами з загальною та розподіленою пам'яттю. Перші набагато легше програмувати, залишаючись загалом в звичній фон-нейманівській парадигмі, але вони обмежені в масштабуванні на нарощуванні кількості процесорів. З іншого боку, архітектури з розподіленою пам'яттю програмувати досить важко, але потенціал для нарощування швидкодії для них є набагато більшим.

NUMA є певним компромісом між паралельними архітектурами з загальною та розподіленою пам'яттю. Перші набагато легше програмувати, залишаючись загалом в звичній фон-нейманівській парадигмі, але вони обмежені в масштабуванні на нарощуванні кількості процесорів. З іншого боку, архітектури з розподіленою пам'яттю програмувати досить важко, але потенціал для нарощування швидкодії для них є набагато більшим.

Склад: однорідні базові модулі (комірки) з невеликого числа процесорів і блоку пам'яті. Модулі об'єднані за допомогою високошвидкісного комутатора. Підтримується єдиний адресний простір, апаратно підтримується доступ до віддаленої пам'яті. Доступ до локальної пам'яті в кілька разів швидший, ніж до віддаленої. У випадку апаратного підтримання когерентності кешів у всій системі говорять про архітектуру cc — NUMA (cache — coherent NUMA)

2.3 Паралельне обчислення інструкцій програмного коду

Поява багатоядерних процесорів є початком нової ери обчислень. Компанії, що виробляють процесори, практично підійшли до порогу підвищення продуктивності обчислень. Застарілий спосіб підвищення продуктивності шляхом збільшення тактової частоти призводить до зростання енергоспоживання. Перехід на всі більш тонкі технології виробництва мікросхем загострює проблему витоку електричного струму (при скороченні розмірів транзисторів її ймовірність зростає) і нагріву самого процесора. Тому вже в перше десятиліття XXI ст. з'явилася необхідність у розвитку методів, що підвищують продуктивність процесора в результаті збільшення кількості обчислювальних ядер в одному корпусі та інструкцій програмного коду, виконуваних за один такт.

Для вирішення проблеми збільшення продуктивності процесорів було запропоновано реалізувати паралельне виконання на рівні інструкцій (Instruction Level Parallelism - ILP) або потоків (Thread Level Parallelism - TLP) [36]. Розпаралелювання на рівні потоків TLP, на відміну від ILP, управляється програмно. Віртуальна багатопоточність створюється в результаті виділення в одному фізичному процесорі двох або більше логічних процесорів. Класичним прикладом такого підходу стала технологія Hyper-Threading (HT) компанії Intel [37]. Внаслідок того, що протягом такту, як правило, не всі виконавчі модулі процесора задіяні, їх можна завантажити паралельним потоком завдань. Зрозуміло, що вдвічі продуктивність не збільшиться, оскільки паралельні потоки використовують загальні пам'ять, кеш і т.д. (до того ж виникають втрати через синхронізації і розпаралелювання інструкцій), але вона в цілому зростає на 15-30%. Мінусом використання технології TLP є виникнення конфліктів, коли одному потоку потрібні результати виконання іншого договору, що призводить до очікування і зростанню кількості тактів, необхідних для виконання інструкцій [38].

Очевидно, щоб уникнути описаних вище проблем потрібно ізолювати в межах одного процесора виконання різних потоків інструкцій. Причому кожен з них повинен посилати команди на своє ядро, тобто для реалізації процесу паралельного виконання завдань слід інтегрувати два ядра або більше в одному центральному процесорі. Крім іншого, така многопроцессорная конфігурація на одному кристалі забезпечує більш високу швидкість обміну між ядрами, ніж використання зовнішніх шин, комутаторів тощо

2.4 Програмне забезпечення багатоядерних систем

Розвиток багатоядерних систем - це шлях до повсюдного використання паралельних обчислень. Як відомо, найбільш поширеним способом підвищення продуктивності є саме розпаралелювання потоку команд або потоку даних. Розпаралелювання даних - це застосування однієї операції відразу до кількох елементів масиву даних. Паралелізм завдань передбачає розбиття обчислювального процесу на кілька підзадач (процесів, потоків), кожна з яких виконується на своєму ядрі (процесорі). Багатоядерні системи відносять до класу MIMD (Multiple Instruction, Multiple Data) [39]. У них кілька програмних гілок виконуються одночасно і незалежно, але в певні моменти вони обмінюються даними.

Активне впровадження багатоядерних систем передбачає істотну зміну стилю програмування: розробники будуть змушені використовувати паралельні потоки, породження і обробку асинхронних подій і ін. Іншими словами, нова апаратна архітектура вимагає зміни програмної парадигми - переходу від послідовного стилю програмування до паралельного.

2.5.1 Стандарти паралельного програмування

При розробці паралельних програм використовуються спеціалізовані бібліотеки і системи паралельного програмування PVM, LAM, CHMP і ін [40]. Три основні підходи до реалізації цих систем розрізняються методами взаємодії паралельних завдань. Перший підхід базується на концепції обміну повідомленнями, другий - на використанні роздільної пам'яті, третій спирається на стандарт POSIX і об’єднує ці два підходи.

Найбільш відомим представником першої групи є специфікація MPI (Message Passing Interface) для мов програмування C та Фортран, перший варіант якої з'явився в 1994 році [41]. MPI забезпечує приблизно 200 функцій, охоплює безліч компіляторів і операційних систем. Серед найбільш поширених її реалізацій - бібліотека MPICH. Крім того, пропонуються кілька комерційних реалізацій MPI, наприклад MPI / Pro компанії Verari Systems Software. MPI / Pro оптимізує час роботи паралельних програм і підтримує їх масштабованість за рахунок балансування параметрів продуктивності та використання ресурсів. Verari пропонує версії MPI / Pro для різних операційних систем, в тому числі Windows, Linux, Mac OS X, LynxOS, і таких комунікаційних середовищ, як Gigabit Ethernet, Myrinet і InfiniBand.

До другої групи відноситься специфікація OpenMP (Open specifications for Multi-Processing) [42]. Її перша версія (www.openmp.org), яка була випущена в 1997 році, призначалася для мови Фортран. До появи OpenMP «доклали руку» компанії IBM, Intel, Sun Microsystems і Hewlett-Packard. У 1998 році були створені варіанти OpenMP для мов

програмування C / C ++, а останній є версія 2.5. Підтримка специфікації OpenMP забезпечена у всіх компіляторах Intel починаючи з шостої версії, в Microsoft Ci / C ++ починаючи з Visual Studio 2005, а також в GCC.

OpenMP - це набір спеціальних директив компілятору (pragma), бібліотечних функцій і змінних середовища. Найбільш оригінальні директиви компілятору, які використовуються для позначення областей в коді і можуть виконуватися паралельно. Компілятор, що підтримує OpenMP, перетворює вихідний код і вставляє відповідні виклики функцій для паралельного виконання цих областей коду.

У третю групу входить специфікація POSIX (Portable Operating System interface for unIX), перший опис якої було опубліковано в 1986 році (www.pasc.org) [43]. Основна специфікація розроблена як IEEE 1003.1 і схвалена як міжнародний стандарт ISO / IEC 9945-1: 1990. З точки зору організації паралельних обчислень найбільший інтерес представляють три частини стандарту - 1003.1a (OS Definition), 1003.1b (Realtime Extensions) і 1003.1c (Threads). В рамках POSIX можна реалізацію паралельних обчислень на основі обміну повідомленнями (аналогічно MPI) або розділяється пам'яті (як в OpenMP). Природно, в POSIX допустима і будь-яка комбінація цих методів. Найбільшою мірою стандарту POSIX відповідають (і відповідним чином сертифіковані) операційні системи реального часу LynxOS і Integrity.

2.5.2 Підтримка багатоядерності на рівні операційної системи

Багатоядерні процесори зажадають від операційних систем підтримки різних архітектур багатопроцесорної обробки [44]. Компанія QNX Software Systems обюявіла про випуск комплекту розробника QNX Momentics Multi-Core Edition. Цей набір інструментів призначений для створення програмного забезпечення та його міграції на багатоядерні апаратні рішення нового покоління, в тому числі процесори BCM12xx і BCM14xx компанії Broadcom, процесор MPC8641D компанії Freescale і багатоядерні процесори Intel. Буде надано підтримку проектам кілька моделей многопроцессорности для багатоядерних архітектур: асиметрична AMP (забезпечення повного управління та відмовостійкості); симетрична SMP (максимальні паралелізм і масштабованість); «Виняткова» BMP (підтримка міграції коду і зниження складності розробки).

Підтримку багатоядерних систем на базі процесорів AMD64, Sun UltraSPARC T1 і Intel забезпечує ОС Solaris 10. Наприклад, вбудована система віртуалізації і захисту інформації Solaris Containers дозволяє системному адміністратору організовувати в рамках єдиної операційної системи кілька віртуальних системних розділів - «зон». Кожній зоні допустимо призначити свій контейнер - набір локалізованих системних ресурсів. Контейнери можуть

служити основою для управління ресурсами на рівні ядер. Реалізовані в Solaris 10 функції так званого «прогнозованого самовідновлення» (Predictive Self-Healing) забезпечують автоматичне визначення збоїв в роботі ядер і їх переклад в пасивний режим без впливу на роботу інших ядер процесора. Підтримка багатоядерних систем реалізована в деяких дистрибутивах ОС Linux, наприклад Red Hat Enterprise Linux 7.

2.5.3 Багаторівнева віртуалізація

Поява багатоядерних процесорів дасть потужний додатковий поштовх масовому впровадженню технологій віртуалізації. Назвемо деякі з відомих підходів.

ARINC-653 (Avionics Application Software Standard Interface) [45]. Стандартний інтерфейс, розроблений компанією ARINC в 1997 році, вводить концепцію ізольованих розділів на основі універсального програмного інтерфейсу APEX (Application / Executive) між операційною системою і прикладним програмним забезпеченням. Вимоги інтерфейсу визначені так, щоб дозволити програмам контролювати диспетчеризацію, зв'язок і стан внутрішніх оброблюваних елементів.

У 2003 році прийнята нова редакція ARINC-653, в якій введена концепція ізольованих віртуальних машин. Її особливістю є жорстке і заздалегідь визначений квантування часу між віртуальними машинами, а метою - забезпечення гарантій того, що чи не виникнуть спільні відмови системи. Стандарт ARINC-653 реалізований для операційних систем реального часу LynxOS-178, VxWorks, Integrity, CsLeos і ін.

2.6 Інтерфейс передачі повідомлень (MPI)

MPI (англ. Message Passing Interface, інтерфейс передачі повідомлень) - це специфікація, розроблена в 1993-1994 роках групою MPI Forum, і забезпечує реалізацію моделі обміну повідомленнями між процесами [46]. Остання версія цієї специфікації: MPI-2. У моделі програмування MPI програма породжує кілька процесів, взаємодіючих між собою за допомогою виклику підпрограм прийому і передачі повідомлень [47].

Зазвичай, при ініціалізації MPI-програми створюється закріплений набір процесів, причому (що, втім, не обов'язково) кожен з них виконується на своєму процесорі. У цих процесах можуть виконуватися неоднакові програми, тому MPI-модель іноді називають MIMD-моделлю (Multiple Instruction, Multiple Data), на відміну від SIMD моделі, де на кожному процесорі виконуються тільки однакові завдання. MPI підтримує двох точках і глобальні,

синхронні і асинхронні, блокуючі і Неблокуючий типи комунікацій. Приватний механізм - комунікатор - ховає від програміста внутрішні комунікаційні структури. Будова комунікацій може модифікуватися протягом часу життя процесу, але число завдань має залишатися незмінною (MPI-2 підтримує динамічне зміну кількості завдань).

Специфікація MPI забезпечує переносимість програм на рівні вихідних кодів [47]. Підтримується робота на гетерогенних кластерах і симетричних мультипроцесорних системах. Чи не підтримується запуск процесів під час виконання MPI-програми. У специфікації відсутні опис паралельного введення-виведення і налагодження програм - ці можливості можуть бути включені до складу конкретної реалізації MPI у вигляді додаткових пакетів або утиліт. Сумісність різних реалізацій не гарантоване.

Важливою властивістю паралельної програми є детермінізм - програма має завжди давати однаковий результат для однакових наборів вхідних даних. Модель передачі повідомлень в загальному такого властивості немає, оскільки не визначений порядок отримання повідомлень від двох процесів третім. Якщо ж один процес послідовно посилає кілька повідомлень іншому процесу, MPI гарантує, що одержувач отримає їх саме в тому порядку, в якому вони були передані. Відповідальність за забезпечення детермінованого виконання програми покладається на програміста [47].

Версія MPI 2.1 вийшла на початку вересня 2008 року. Версія MPI 2.2 вийшла на 4 вересня 2009 року версія MPI 3.0 вийшла 21 вересня 2012 року.

2.7 Висновки до другого розділу

За результатами дослідження другого розділу сформовано наступні висновки. У розділі було проаналізовано різні архітектури операційних систем реального часу і обрано архітектуру «Клієнт-сервер». Переваги такої архітектури:

– підвищена надійність, так як кожен сервіс є, по суті, самостійним додатком і його потрібно легше налагодити і відстежити помилки;

– покращена масштабованість, оскільки непотрібні сервіси можуть бути виключені з системи без шкоди до її працездатності;

– підвищена відмовостійкість, так як сервіс, який завис може бути перезапущений без перезавантаження системи.

Було досліджено основні функції диспетчера у ОСРЧ для подальшого проектування власного диспетчера. Розглянуто специфікацію MPI завдяки якій можливо більш раціонально використовувати ресурси мікропроцесора, а саме, кожного з його ядер. Використання новітніх мікропроцесорів забезпечить швидкодію при обробці даних з датчиків, завдяки близькому розташуванню ядер та їх кешей, у яких можливо швидко зберігати необхідні дані. Також дає можливість запускати три версії трьохверсійної системи одночасно, що забезпечить більшу швидкість обчислення і, отже, збільшить відмовостійкість.

3 РОЗРОБКА ДИСПЕТЧЕРА ОПЕРАЦІЙНОЇ СИСТЕМИ РЕАЛЬНОГО ЧАСУ

Згідно з технічним завданням даний розділ присвячується розробці програмного забезпечення - диспетчера ОСРЧ для забеспечення трьохверсійного програмування на базі одного багатоядерного процесора. В даному розділі розглянуто структуру модель диспетчера, алгоритм виконання циклу ПЗ та засоби, за допомогою яких він запрограмований.

3.1 Призначення розробленного диспетчера

На сьогодні у підприємстві ПрАТ «СНПО «Импульс» ведеться розробка нового промислового контролера МСКУ-5 на базі багатоядерного мікропроцесора.

У системах управління на АЕС зазвичай використовують три окремі мікроконтролери.

Якщо використовувати усі чотири ядра кожного мікроконтролера, це збільшить відмовостійкість системи. Головним нововведенням є трьохверсійне відмовостійке програмування на базі новітніх чотирьохядерних мікроконтролерів Intel із кожною версією на одне з ядер. Тому, впровадження мультиверсійності в новітніх мікроконтролерах є запорукою стабільного технологічного процесу.

3.2 Розробка диспетчера ОСРВ

Програма-диспетчер (англ. dispatcher, supervisor (program), monitor (program)) — вбудоване або факультативне ПЗ операційної системи що керує чи показує виконання інших програм чи завдань. Програма-диспетчер може використовуватись для координації та взаємодії периферійних пристроїв цифрової обчислювальної машини чи кількох машин, об'єднаних в обчислювальну систему.

Робота диспетчер трьохверсійної мультипроцесорної обчислювальної системи демонструється на прикладі реалізації ФБ (рис. 3.1).

До операційної системи входить диспетчер ФБ. Диспетчер ФБ запускає паралельно три функціонально ідентичних ФБ. Для очікування закінчення виконання трьох ФБ використовується команда типу «Multiple wait» (очікування декількох подій). Диспетчер ФБ визначає мажоритарним методом «2 із 3» вірний результат. У разі невідповідності результату одного ФБ відповідне повідомлення передається підсистемі верхнього рівня (для повідомлення оператору, архівації та ін.). Вірний результат виконання записується в базу даних (БД), процес триває, на відміну від двухверсійного варіанту, коли система управління повинна перейти в

безпечний стан.

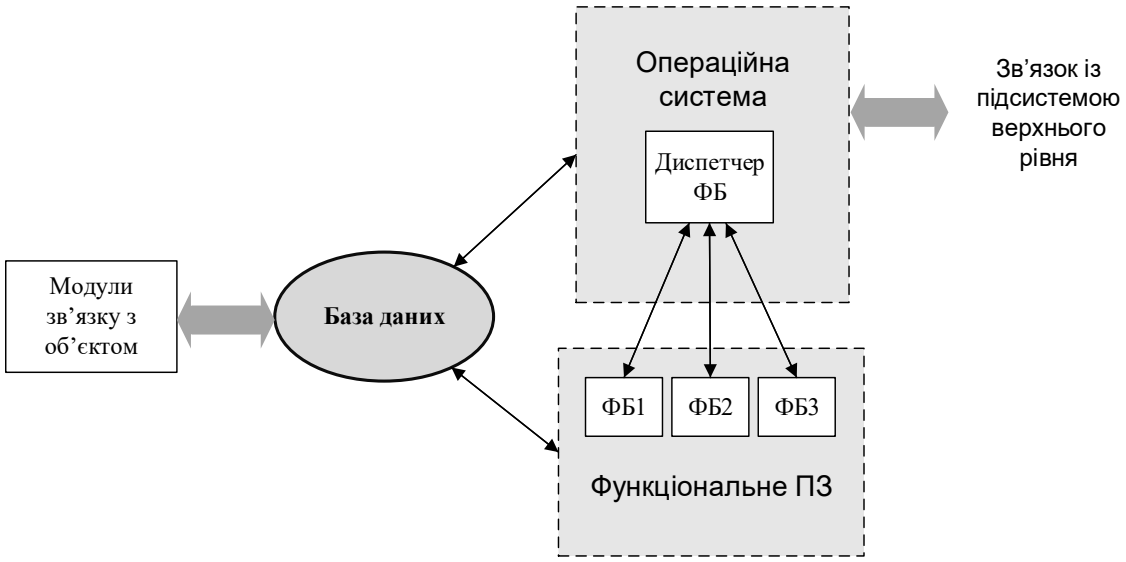


Рисунок 3.1 – Структура ПЗ

Виконуваний модуль ФБ являє собою фрагмент коду, який запускається диспетчером переходом по таблиці ФБ згідно з номером блоку - першим словом структури поточної викликаючої послідовності. Старший байт слова - номер групи - відповідає функціональному призначенню диспетчера, молодший - вказує безпосередньо на місце адреси модуля ФБ в таблиці.

Алгоритм виконання циклу ПЗ наступний (рис. 3.2):

– запуск трьох модулів ФБ в мікроконтролері;
– виконання модулів ФБ;
– формування результату по принципу мажорірованія «2 із 3»;
– сповіщення оператора, якщо на одному з ланок знайдено помилку;
– запис результату в БД;
– установ покажчика в списку викликів на наступний ФБ.

Рисунок 3.2 – Виконання циклу ПЗ

3.3 Програмування диспетчера

Наявність потужної обчислювальної бази не гарантує факту ефективного її використання. Швидкодія паралельної системи значною мірою залежить від обраної архітектури, вдало розробленого алгоритму, правильної декомпозиції поставленої задачі та ряду інших факторів. Тому розробка продуктивного та ефективного паралельного рішення являється доволі нетривіальною задачею, тим паче в час масового використання багатоядерних мікропроцесорів.

3.3.1 Диспетчеризація потоків

Для того, щоб орієнтуватися у архітерктурі «event» ввів такі сутності:

– подія (повідомлення): може ініціюватися за таймером або безпосередньо, викликом відповідної функції;

– обробник події: функція, яка викликається при виникненні події. На одну подію може бути повішено кілька обробників і вони будуть викликатися послідовно;

– таймер: можна включити таймер, після закінчення якого буде одноразово генеруватися заданий подія;

– чергу повідомлень: кільцевої буфер повідомлень FILO.

– диспетчер таймерів: функція, що викликається по перериванню таймера;

– диспетчер подій: функція, яка крутиться в головному циклі.

Програмування головного таймера (рис. 3.3), який викликає переривання, що повідомляє процесору про настання події. Макросом ISR в avr-gcc описується обробник переривання. Ця функція викликається 4096 раз в секунду. А наш диспетчер таймерів викликається один раз в 10 мсек (з невеликою похибкою).

Бажаючі можуть використовувати вбудований таймер або інший період таймера, суть від цього не змінюється (рис. 3.3).

```
unsigned int calc_delay;
ISR(INT0_vect) {
  if (calc_delay > 41) {
    calc_delay = 0;
    dispatchTimer();
  } else
    calc_delay++;
}
```

Рисунок 3.3 Головний лічильник

Виставляємо обробник і зводимо таймер. Далі вже працює система - відстежує таймер і створює подія. Події обробляються в порядку надходження (рис. 3.4). Функція-обробник виконує дії і знову зводить таймер (якщо треба).

```
setHandler(MSG_LCD_REFRESH, &refreshLCD);
setTimer(MSG_LCD_REFRESH, 0, 20);

setHandler(MSG_ADC_CYCLE, &readKey);
setHandler(MSG_KEY_REPEAT, &repeatKey);

setHandler(MSG_KEY_PRESS, &analyseKey);

setHandler(MSG_1W_TEMP, &owTemp);
setTimer(MSG_1W_TEMP, 1, 10);
```

Рисунок 3.4 Процедура ініціалізації

3.3.3 Діапазон функцій

У проекті використано три моделі міжзадачної взаємодії і синхронізації:

– Сервіси прив'язані до завдань: ОСРВ наділяє завдання атрибутами, які забезпечують взаємодію між ними. Як приклад розглянемо сигнали.

– Об'єкти ядра - автономні засоби зв'язку або синхронізації. Приклади: прапори подій, поштові ящики, черги / канали, семафори і мьютекс.

– Обмін повідомленнями - раціоналізувати схема, в якій ОСРВ дозволяє створювати об'єкти повідомлень і здійснювати їх передачу від однієї задачі до іншої або кількох. Це має основоположне значення для архітектури ядра, і тому така система називається «ОСРВ з обміном повідомленнями».

Механізми, які ідеально підходять для різних процесів, будуть відрізнятися. Їх можливості можуть перетинатися, тому варто задуматися про масштабованості цих моделей. Наприклад, коли у програмі є кілька черг, але тільки один поштовий ящик, можна реалізувати поштову скриньку з чергою на один елемент. Цей об'єкт буде не зовсім оптимальним, але весь код роботи поштового ящика не буде включений в програму, і, отже, масштабованість зменшить обсяг займаної пам'яті ОСРВ.

3.3.4 Загальні змінні або області пам'яті

Спрощений підхід до взаємодії між завданнями - це наявність в системі змінних або областей пам'яті, які доступні для всіх завдань. Якщо змінна є просто байтом, то запис в ній або читання з неї, ймовірно, буде атомарної операцією (тобто безперервної), але необхідно проявляти обережність, якщо процесор дозволяє інші операції на байтах пам'яті, оскільки вони можуть бути переривати і може виникнути проблема синхронізації. Один із способів реалізації блокування / розблокування - відключити переривання на короткий час.

Якщо використовується область пам'яті, все одно потрібна блокування. Допускається використання першого байта в якості блокуючого прапора, з урахуванням, що архітектура пам'яті надає атомарний доступ до цього байту. Одна задача завантажує дані в область пам'яті, встановлює прапор і потім чекає скинути його. Інше завдання чекає установки прапора, виконує читання даних і скидає прапор. Використання відключення переривання в якості блокування менш розумно, так як переміщення всього буфера даних може зайняти деякий час.

Таке використання загальної пам'яті схоже на реалізацію багатьох міжпроцесорних засобів зв'язку в багатоядерних системах. У деяких випадках апаратне блокування і / або

переривання вбудовані в міжпроцесорний інтерфейс загальної пам'яті.

3.3.5 Сигнали

Використовувати сигнали є одним з найпростіших механізмів взаємодії між завданнями, пропонованих традиційними ОСРВ. Вони містять набір бітових прапорів (8, 16 або 32, в залежності від конкретного застосування), який пов'язаний з конкретним завданням.

Прапор сигналу (або кілька прапорів) може бути встановлений будь-яким завданням за допомогою логічної операції «АБО». Прапор (прапори) може прочитати тільки завдання, яка містить сигнал. Процес читання, як правило, деструктивний, тобто прапори також скидаються.

У деяких системах сигнали реалізуються більш складним способом, так що ця функція призначена завданням-власником сигналу, автоматично виконується, коли встановлюються будь-які сигнальні прапори (рис. 3.5). Це усуває необхідність того, щоб завдання контролювала прапори сама. Це дещо схоже на обробника переривань.

```
struct sigaction act;
memset(&act, 0, sizeof(act));
act.sa_handler = hdl;
sigset_t   set;
sigemptyset(&set);
sigaddset(&set, SIGUSR1);
sigaddset(&set, SIGUSR2);
act.sa_mask = set;
sigaction(SIGUSR1, &act, 0);
sigaction(SIGUSR2, &act, 0);
```

Рисунок 3.5 Використання сигналів

3.3.6 Групи прапорів подій

Групи прапорів подій схожі на сигнали тим, що є біт-орієнтованим засобом для взаємодії між завданнями. Аналогічним чином вони можуть містити 8, 16 або 32 біта. На відміну від сигналів, вони є незалежними об'єктами ядра і не «належать» до будь-якої конкретної задачі. Будь-яке завдання може встановлювати і скидати прапори подій, використовуючи логічні операції «АБО» і «І». Таким же чином будь задача може перевірити прапори подій з використанням тих же операцій. У багатьох ОСРВ можна зробити блокуючий виклик API для комбінації прапорів подій. Тобто завдання може бути припинена до тих пір, поки не буде встановлена конкретна комбінація прапорів подій. Також може бути доступна опція «consume» при перевірці прапорів подій (рис. 3.6), яка скидає все прапори.

```
template<class ...TParams>
class AbstractEventHandler
{
public:
virtual void call( TParams... params ) = 0;
consume:
AbstractEventHandler() {}
};
```

Рисунок 3.6 Подія

3.3.7 Семафори

Семафори є незалежними об'єктами ядра, що використовуються для обліку ресурсів. Існує два типи семафорів: виконавчі (може мати тільки два значення) і загальні (необмежене число значень). Деякі процесори підтримують (атомарні) інструкції, які полегшують швидку реалізацію довічних семафорів (рис. 3.7). Двійкові семафори можуть бути реалізовані як загальні семафори зі значенням 1.

```
class AutoResetEvent
{
private:
    // m_status == 1: Event object is signaled.
    // m_status == 0: Event object is reset and no threads are waiting.
    // m_status == -N: Event object is reset and N threads are waiting.
    std::atomic<int> m_status;
    Semaphore m_sema;
```

Рисунок 3.7 Семафор

Будь-яке завдання може спробувати привласнити семафор, щоб отримати доступ до ресурсу. Якщо поточне значення семафора більше 0 (семафор вільний), значення лічильника зменшується на 1, отже, привласнення буде успішним. У багатьох ОС можна застосувати блокуючий механізм для присвоєння семафора. Це означає, що завдання може перебувати в стані очікування до тих пір, поки семафор не буде звільнений іншим завданням. Будь-яке завдання може звільнити семафор, і тоді значення семафора збільшиться.

3.3.8 Черги

Черги - незалежні об'єкти ядра, що надають механізм для передачі повідомлень. Вони трохи більш гнучкі і складні, ніж поштові скриньки. Розмір повідомлення залежить від реалізації, але зазвичай він фіксований і орієнтований на слово / покажчик.

Будь-яке завдання може відправляти повідомлення в чергу, і це може повторюватися до тих пір, поки черга не заповниться, після чого будь-які спроби відправки приведуть до помилки. Довжина черги зазвичай визначається користувачем при її створенні або налаштування системи. У багатьох ОСРВ можна застосувати блокуючий механізм для відправки в чергу. Тобто якщо чергу заповнена, завдання може бути припинена до тих пір, поки повідомлення в черги не буде прочитано іншим завданням. Будь-яке завдання може зчитувати повідомлення з черги. Повідомлення читаються в тому ж порядку, в якому вони були відправлені (First in - First out, FIFO). Якщо завдання намагається прочитати з порожньою черзі, вона отримає відповідь про помилку. У багатьох ОСРВ можна застосувати блокуючий механізм для читання з порожньою черзі. Тобто якщо чергу порожня, завдання може бути припинена до тих пір, поки сполучення не буде відправлено в чергу іншим завданням.

Швидше за все в ОСРВ буде механізм для відправки повідомлення в початок черги, це називається «jamming». Деякі ОСРВ також підтримують функцію «broadcast». Це дозволяє відправляти повідомлення будь-кому завданням, припинених при читанні черзі.

Крім того, ОСРВ може підтримувати відправку і читання повідомлень змінної довжини. Це дає велику гнучкість, але тягне за собою додаткові накладні витрати.

Багато ОСРВ підтримують інший тип об'єкту ядра - «канал» ("pipes"). По суті, канал аналогічний черзі, але обробляє байт-орієнтовані дані.

Функціональність черг не представляє інтересу, але слід розуміти, що вони мають більше накладних витрат на пам'ять і час виконання, ніж поштові скриньки, перш за все тому, що необхідно зберегти два покажчика: початок і кінець черги.

3.3.9 М'ютекси

М'ютекси (взаємовиключні семафори) - незалежні об'єкти ядра, які ведуть себе дуже схожим на звичайні двійкові семафори чином. Вони трохи складніше семафорів і включають концепцію тимчасового володіння (ресурсу, доступ до якого контролюється). Якщо завдання привласнює м'ютекс, тільки ця ж задача може його знову звільнити: м'ютекс (і, отже, ресурс) тимчасово належить завданню (рис. 3.8).

```
void do_sth_1_eq( ) throw()
{
  if (WaitForSingleObject(g_mutex.get(), INFINITE) == WAIT_OBJECT_0)
  {
    g_common_cnt_ex = 0;
    g_common_cnt = 0;
    ReleaseMutex(g_mutex.get());
  }
  else
  {
    assert(0);
  }
}
```

Рисунок 3.8 Використання м’ютексів

М'ютекси підтримуються не всіма ОСРВ, але регулярний двійковий семафор досить просто адаптувати. Необхідно написати функцію «mutex obtain», яка присвоює семафор і призначає ідентифікатор завдання. Потім додаткова функція «mutex release» перевіряє ідентифікатор викликає завдання і звільняє семафор тільки в тому випадку, якщо вона відповідає збереженому значенням, інакше поверне помилку.

3.4 Формування результату по мажоритарному принципу

Реалізація надійного обчислювального процесу розглянуто на прикладі програмного забезпечення, що виконується в контролері з чотирьохядерним процесором по схемі «2 із 3». Вибір схеми обробки по схемі «2 із 3» обумовлений наступним факторами: схеми типу «1 із N» (N=1,2,…) не відповідають вимогам відповідних стандартів і не можуть застосовуватися в системах управління реального часу (не володіють функціональною безпекою).

Схеми типу «N із N» (N = 1,2, ...) не мають великої надійності. Наприклад, користуючись, можна розрахувати, що в схемі «2 із 2» при ймовірності появи прихованого дефекту в циклі управління $p=10^{-7}$ в одній з двох гілок програми середній час напрацювання на відмову системи управління рівне Tср~14 h при часу циклу t = 0,01 s.

Схема «2 із 3» при тій же ймовірності появи прихованого дефекту забезпечує середній час напрацювання на відмову Tср~5285 років. Застосування схеми «2 із 3» (на відміну від схеми «2 із 2») дозволяє також однозначно визначати гілку програми з проявленим дефектом і міняти результат виконання цієї гілки на вірний (рис. 3.9). Застосування схем «2 із N» (N = 4,5, ...) недоцільно, тому що на практиці не потрібно такі наднадійні системи і вони складні в реалізації.

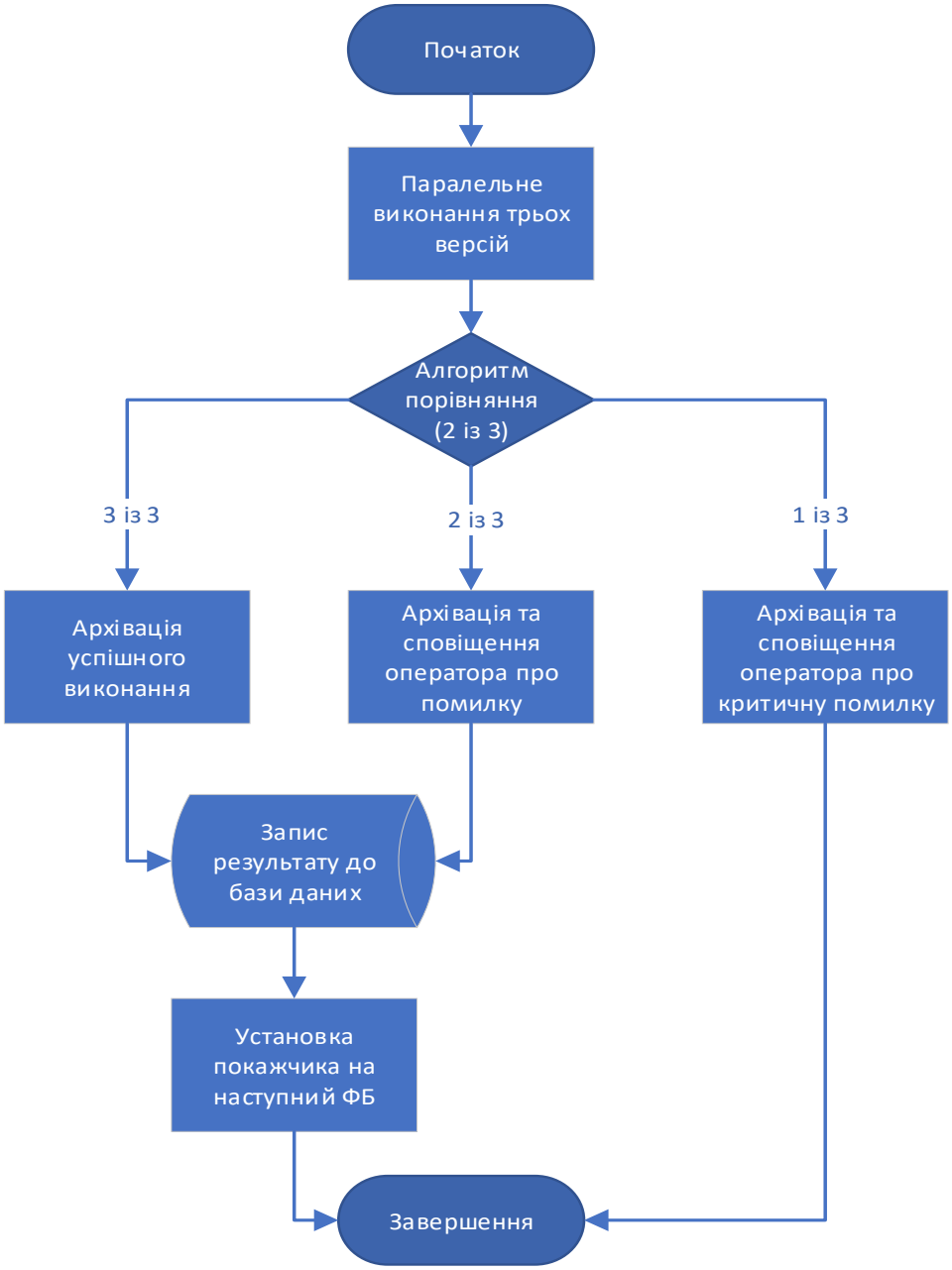


Рисунок 3.9 Блок-схема порівняння результатів по принципу «2 із 3»

Якщо на виході отримуємо три різні результати система повідомляє про критичну помилку і зупиняє роботу системи.

3.5 Висновки до третього розділу

У третьому розділі розглянуто спосіб практичної реалізації диспетчера керуючого обчислювального процесу в багатоядерному мікропроцесорі промислового контролера. Даний спосіб дозволяє знизити ризики відмов від прихованих помилок, оскільки в разі проектних дефектів зменшується ймовірність одночасної і однотипної відмови різних версій ПЗ. При застосуванні даного методу також підвищується відмовостійкість ПЗ за рахунок реалізації принципу мажорірованія «2 із 3» и підвищується коефіцієнт готовності керуючої системи. У системах управління з високими вимогами до надійності розглянутий спосіб може застосовуватися разом з традиційними способами резервування апаратури, наприклад, в контролерах з трьома мікропроцесорами.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ. ЕКОЛОГІЯ

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів, причин пожеж. Розглянуті заходи, які дозволяють забезпечити гігієну праці і виробничу санітарію. На підставі аналізу розроблені заходи з техніки безпеки та рекомендації з пожежної профілактики.

Завданням даної роботи магістра було розроблення програмного забезпечення - диспетчер для забеспечення трьохверсійного програмування на базі одного багатоядерного процесора, і як результат було створено даний проект. За цим проектом в подальшому розроблятиметься реальна система, яка значно полегшить процес розробки трьохверсійної мутипроцесорної системи. Так як в процесі проектування використовувалося електричне обладнання, то аналіз потенційно небезпечних і шкідливих виробничих чинників виконується для персонального комп'ютера на якому буде розроблена дипломна робота.

4.1 Правові та організаційні основи охорони праці

Основним організаційним напрямом у здійсненні управління в сфері охорони праці є усвідомлення пріоритету безпеки праці і підвищення соціальної відповідальності держави, і особистої відповідальності працівників.

На законодавчому рівні визначено такі пріоритетні напрямки з безпеки праці:

– кожен працівник несе безпосередню відповідальність за порушення зазначених Законом, нормами і правилами вимог;

– напрямки реалізації конституційного права громадян на їх життя і здоров’я в процесі трудової діяльності:

– пріоритет життя і здоров’я працівників по відношенню до результатів виробничої діяльності підприємства;

– соціальний захист працівників, повне відшкодування збитків особам, які потерпіли від нещасних випадків на виробництві та професійних захворювань;

– комплексне розв’язання завдань охорони праці;

– підвищення рівня промислової безпеки шляхом забезпечення суцільного технічного контролю за станом виробництв, технологій та продукції, а також сприяння підприємствам у створенні безпечних та нешкідливих умов праці;

– соціальний захист працівників, повне відшкодування збитків особам, які потерпіли від нещасних випадків на виробництві та професійних захворювань;

– використання економічних методів управління охороною праці, участь держави у фінансуванні заходів щодо охорони праці;

Наявні трудові відносини між працівниками і роботодавцями в Україні за темою дипломного проекту регулюються Кодексом законів про працю (КЗпП) України, відповідно до якого права працюючої людини на охорону праці охороняються всебічно та норми охорони праці неухильно інтегровані до правил внутрішнього розпорядку організації/підприємства [48].

4.2 Вимоги до приміщень

Розмір мого робочого місця становить 10 кв. м. Так як розмір площі для одного робочого місця оператора персонального комп'ютера має бути не менше 6 кв. м, а об'єм — не менше 20 куб. м, отже, дане приміщення цілком відповідає зазначеним нормам.

Для зручності спільної роботи з іншими працівниками (обговорення ідей, з'ясування проблем і т.д.) в кімнаті є дивани і журнальний стіл, обставлені живими квітами. Також робочий процес пов'язаний з багатьма документами, теками, журналами для чого приміщення облаштоване принтером і шафою для зручності. Задля дотримання визначеного рівня мікроклімату в будівлі встановлено систему опалення та кондиціонування.

Для забезпечення потрібного рівного освітленості кімната має вікно та систему загального рівномірного освітлення, що встановлена на стелі. Для дотримання вимог пожежної безпеки встановлено порошковий вогнегасник та систему автоматичної пожежної сигналізації.

4.3 Вимоги до організації місця праці

Робочий стіл на досліджуваному місці також містить достатньо простору для ніг. Крісло, що використовується в якості робочого сидіння, є підйомно поворотним, має підлокітники і можливість регулювання за висотою і кутом нахилу спинки, також воно м'яке і виконане з екологічної шкіри, що дає можливість працювати у комфорті. Екран монітору знаходиться на відстані 0.8 м, клавіатура має можливість регулювання кута нахилу 5-15°. Отже, за всіма параметрами робоче місце відповідає нормативним вимогам. Приміщення кабінету знаходиться на другому поверсі дев'яти поверхової будівлі і має об'єм 78 $м^3$, площу – 24 $м^2$. У цьому кабінеті обладнано одне місця праці [49].

Температура в приміщенні протягом року коливається у межах 18–24°C, відносна вологість — близько 50%. Швидкість руху повітря не перевищує 0,2 м/с. Шум в лабораторії

знаходиться на рівні 50 дБА. Система вентилювання приміщення — природна неорганізована, а опалення — централізоване.

Розміщення вікон забезпечує природне освітлення з коефіцієнтом природного освітлення не менше 1,5%, а загальне штучне освітлення, яке здійснюється за допомогою восьми люмінесцентних ламп, забезпечує рівень освітленості не менше 200 Лк.

У кабінеті є електрична мережа з напругою 220 В, яка створює небезпеку ураження електричним струмом. ПК та периферійні пристрої можуть бути джерелами електромагнітних випромінювань, аерозолів та шкідливих речовин (часток тонеру, оксидів нітрогену та озону).

За ступенем пожежної безпеки приміщення належить до категорії В. Кабінет оснащений переносним вуглекислотним вогнегасником ВВК-5.

Наявна аптечка для надання долікарської допомоги, а також у кабінеті роблять вологе прибирання та щоденно провітрюють приміщення.

4.4 Виробнича санітарія

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки можуть бути надалі вирішені питання необхідності забезпечення працюючих достатньою кількістю освітлення, вентиляції повітря, організації заземлення, тощо. [50]

4.5 Мікроклімат

Мікроклімат робочих приміщень – це клімат внутрішнього середовища цих приміщень, що визначається діючої на організм людини з'єднанням температури, вологості, швидкості переміщення повітря. В даному приміщенні проводяться роботи, що виконуються сидячи і не потребують динамічного фізичного напруження, то для нього відповідає категорія робіт Іа. Отже оптимальні значення для температури, відносної вологості й рухливості повітря для зазначеного робочого місця відповідають [50].

Дане приміщення обладнане системами опалення, кондиціонування повітря або припливно-витяжною вентиляцією. У приміщенні на робочому місці забезпечуються оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря [50]. Рівні позитивних і негативних іонів у повітрі мають відповідати. Для забезпечення оптимальних параметрів мікроклімату в приміщенні проводяться перерви в роботі співробітників, з метою його провітрювання. Існують спеціальні системи кондиціонування, які забезпечують підтримання в приміщенні балансу оптимальних параметрів мікроклімату.

Таблиця 5.1 – Норми мікроклімату робочої зони об’єкту

Період року	Категорія робіт	ТемператураC^0	Відносна вологість %	Швидкість руху повітря, м/с
Холодна	легка-1 а	22 - 24	40 – 60	0,1
Тепла	легка-1 а	23 - 25	40 – 60	0,1

Контроль параметрів мікроклімату в холодний і теплий період року здійснюється не менше 3-х разів на зміну (на початку, середині, в кінці).

4.6 Освітлення

Світло є природною умовою існування людини. Воно впливає на стан вищих психічних функцій і фізіологічні процеси в організмі. Хороше освітлення діє тонізуюче, створює гарний настрій, покращує протікання основних процесів вищої нервової діяльності.

Збільшення освітленості сприяє поліпшенню працездатності навіть в тих випадках, коли процес праці практично не залежить від зорового сприйняття. При поганому освітленні людина швидко втомлюється, працює менш продуктивно, виникає потенційна небезпека помилкових дій і нещасних випадків.

Освітленість приміщення має велике значення при роботі на ПЕОМ. Вона багато в чому визначається колірною і мережевий обстановкою. Для зменшеного поглинання світла стеля і стіни вище панелей (1,5-1,7м.). Якщо вони не облицьовані звукопоглинальним матеріалом, фарбуються білою водоемульсійною фарбою (коефіцієнт відбиття повинен бути не менше 0,7). Для забарвлення стіни панелей рекомендується віддавати перевагу світлим фарбам.

Основний потік природного світла при цій повинен бути зліва. Не допускається спрямування основного світлового потоку природного світла праворуч, ззаду і спереду працівника на ПЕОМ.

Робота на ПЕОМ може здійснюватися за таких видах освітлення:

– загальному штучному освітленні, коли відео монітори розташовуються по периметру приміщення або при центральному розташуванні робочих місць у два ряди по довжині кімнати з екранами, звернені в протилежні сторони;

– суміщене освітлення (природне + штучне) тільки при одному і трьох рядном розташуванні робочих місць, коли екран і поверхню робочого столу знаходяться перпендикулярно світла несучій стіні. При цьому штучне освітлення буде виконане стельовими

або підвісними люмінесцентними світильниками, рівномірно розміщеними по стелі рядами паралельно світловим прорізам так, щоб екран відео монітора знаходився в зоні захисного кута світильника, і його проекції не доводилися на екран. Працюючі на ПЕОМ не повинні бачити відображення світильників на екрані. Застосовувати місцеве освітлення при роботі на ПЕОМ не рекомендується.

Природне освітлення, коли робочі місця з ПЕОМ розташовуються в один ряд по довжині приміщення на відстані 0,8 - 1,0 м від стіни з віконними прорізами, і екрани знаходяться перпендикулярно цієї стіни. Основний потік природного світла при цій повинен бути зліва. Не допускається спрямування основного світлового потоку природного світла праворуч, ззаду і спереду працює на ПЕОМ. Оптимальна відстань очей до екрана відео монітора повинна становити 60-70 см, допустиме не менше 50 см. Розглядати інформацію ближче 50 см не рекомендується.

У проекті, що розробляється, передбачається використовувати суміщене освітлення. У світлий час доби використовуватиметься природне освітлення приміщення через віконні отвори, в решту часу використовуватиметься штучне освітлення. Штучне освітлення створюється газорозрядними лампами.

Штучне освітлення в робочому приміщенні передбачається здійснювати з використанням люмінесцентних джерел світла в світильниках загального освітлення, оскільки люмінесцентні лампи мають високу потужність (80 Вт), тривалий термін служби (до 10000 годин), спектральний складом випромінюваного світла, близький до сонячного. При експлуатації ЕОМ виконується зорова робота IVв розряду точності (середня точність). При цьому нормована освітленість на робочому місці (Ен) рівна 200 лк. Джерелом природного освітлення є сонячне світло.

У приміщенні, де розташовані ЕОМ передбачається природне бічне освітлення. Джерелом природного освітлення є сонячне світло. Регулярно повинен проводитися контроль освітленості, який підтверджує, що рівень освітленості задовольняє ДБН і для даного приміщення в світлий час доби достатньо природного освітлення [51].

Розрахунок освітлення.

Для побутових приміщень світловий коефіцієнт приймається не менше за 1/10:

$$S_b = \left(\frac{1}{5} \div \frac{1}{10}\right) \cdot S_n, \tag{4.1}$$

де S_b – площа віконних прорізів, м^2;

$$S_n = a \cdot b = 4 \cdot 6 = 24 \text{ м}^2, \tag{4.2}$$

$$S = 1/8 \cdot 24 = 3 \text{ м}^2.$$

Приймаємо 1 вікно площею S=3 м^2.

Світильники загального освітлення розташовуються над робочими поверхнями в рівномірно-прямокутному порядку. Для організації освітлення в темний час доби передбачається обладнати приміщення, довжина якого складає 5 м, ширина 5 м, світильниками ЛПО2П, оснащеними лампами типу ЛБ (дві по 80 Вт) з світловим потоком 5400 лм кожна.

Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників n виробляється по формулі:

$$n = \frac{E \cdot S \cdot Z \cdot K}{F \cdot U \cdot M}, \tag{4.3}$$

де E – нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S – освітлювана площа, м^2; S = 24 м^2;

Z – поправочний коефіцієнт світильника (Z = 1,15 для ламп розжарювання та ДРЛ; Z = 1,1 для люмінесцентних ламп) приймаємо рівним 1,1;

K – коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації – 1,5;

U – коефіцієнт використання, залежний від типу світильника, показника індексу приміщення і т.п. – 0,575

M – число люмінесцентних ламп в світильнику – 2;

F – світловий потік лампи – 5400лм (для ЛБ-80).

Підставивши числові значення у формулу, отримуємо:

$$\text{n} = \frac{300 * 24 * 1.1 * 1.5}{5400 * 0.575 * 2} \approx 1.9 \tag{4.4}$$

4.7 Вентилювання

У приміщенні, де знаходяться ЕОМ, повітрообмін реалізується за допомогою природної організованої вентиляції (вентиляційні шахти), тобто при V приміщення > 40 м^3 на одного працюючого допускається природна вентиляція. Цей метод забезпечує приток потрібної кількості свіжого повітря, що визначається в СНіП.

Також має здійснюватися провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.8 Заходи з організації виробничого середовища та попередження виникнення надзвичайних ситуацій

Застосовують різні електричні захисні засоби від ураження струмом:

а) Ізолюючі - ізолюють людини від струмоведучих або заземлених частин, а так-же від землі. Вони діляться на основні та додаткові.

б) Основні - володіють ізоляцією, здатної довго витримувати робоче напругу електроустановки і тому ними дозволяється стосуватися струмоведучих частин, знаходячи-трудящих під напругою. До них відносяться: в електроустановках до 1000 Вт - діелектричної рукавички, ізолюючі штанги, ізолюючі і електровимірювальні кліщі і т.д .; понад 1000 Вт - ізолюючі штанги, і електровимірювальні кліщі, а також кошти для ремонтних робіт під напругою понад 1000Вт.

в) Запобіжні - володіють ізоляцією нездатною витримати робоча напруга електроустановки, і тому вони не можуть самостійно захищати людину від ураження струмом під цим напругою. Їх значення - посилити захисні дії основних і ізолюючих засобів, разом з якими вони повинні застосовуватися, при чому при використання основних захисних засобів достатньо застосування одного запобіжного захисного засобу. До запобіжних відносяться засоби в електроустановках до 1000 Вт - діелектричні калоші килимки, а також ізолюючі підставки.

Розрахунок захисного заземлення (забезпечення електробезпеки будівлі).

Згідно з класифікацією приміщень за ступенем небезпеки ураження електричним струмом, приміщення в якому проводяться всі роботи відноситься до першого класу (без підвищеної небезпеки). Під час роботи використовуються електроустановки з напругою живлення 36 В, 220 В, та 360 В. Опір контура заземлення повинен мати не більше 4 Ом.

Розрахунок проводять за допомогою методу коефіцієнта використання (екранування) електродів. Коефіцієнт використання групового заземлювача η – це відношення діючої провідності цього заземлювача до найбільш можливої його провідності за нескінченно великих відстаней між його електродами. Коефіцієнт використання вертикальних заземлювачів $\eta_в$ в залежності від розміщення заземлювачів та їх кількості знаходиться в межах 0,4…0,99. Взаємну екрануючу дію горизонтального заземлювача (з'єднувальної смуги) враховують за допомогою коефіцієнта використання горизонтального заземлювача η_c.

Послідовність розрахунку.

1) Визначається необхідний опір штучних заземлювачів $R_{шт.з.}$:

$$R_{шт.з.} = \frac{R_д \cdot R_{пр.з.}}{R_{пр.з.} - R_д}, \quad (4.5)$$

де $R_{пр.з.}$ – опір природних заземлювачів; $R_д$ – допустимий опір заземлення. Якщо природні заземлювачі відсутні, то $R_{шт.з.}=R_д$.

Підставивши числові значення у формулу, отримуємо:

$$R_{шт.з.} = \frac{4 \cdot 40}{40-4} \approx 4\text{Ом} \quad (4.6)$$

2) Опір заземлення в значній мірі залежить від питомого опору ґрунту ρ, Ом·м. Приблизне значення питомого опору глини приймаємо ρ=40 Ом•м (табличне значення).

3) Розрахунковий питомий опір ґрунту, $\rho_{розр}$, Ом·м, визначається відповідно для вертикальних заземлювачів $\rho_{розр.в}$, і горизонтальних $\rho_{розр.г}$, Ом·м за формулою:

$$\rho_{розр..} = \psi \cdot \rho, \quad (4.7)$$

де ψ – коефіцієнт сезонності для вертикальних заземлювачів I кліматичної зони з нормальною вологістю землі, приймається для вертикальних заземлювачів $\rho_{розр.в}$=1,7 і горизонтальних $\rho_{розр.г}$=5,5 Ом·м.

$$\rho_{розр.в} = 1{,}7 \cdot 40 = 68 \text{ Ом•м} \quad (4.8)$$
$$\rho_{розр.г} = 5{,}5 \cdot 40 = 220 \text{ Ом•м}$$

4) Розраховується опір розтікання струму вертикального заземлювача $R_в$, Ом, за.

$$R_в = \frac{\rho_{розр.в}}{2 \cdot \pi \cdot l_в} \cdot \left(ln\frac{2 \cdot l_в}{d_{ст}} + \frac{1}{2} \cdot ln\frac{4 \cdot t + l_в}{4 \cdot t - l_в} \right), \quad (4.9)$$

де $l_в$ – довжина вертикального заземлювача (для труб - 2−3 м; $l_в$=3 м);

$d_{ст}$ – діаметр стержня (для труб - 0,03−0,05 м; $d_{ст}$=0,05 м);

t – відстань від поверхні землі до середини заземлювача, яка визначається за ф.:

$$t = h_в + \frac{l_в}{2}, \tag{4.10}$$

де $h_в$ – глибина закладання вертикальних заземлювачів (0,8 м); тоді

$$t = 0{,}8 + \frac{3}{2} = 2{,}3 \text{ м} \tag{4.11}$$

$$R_в = \frac{68}{2 \cdot \pi \cdot 3} \cdot \left(ln \frac{2 \cdot 3}{0{,}05} + \frac{1}{2} \cdot ln \frac{4 \cdot 2{,}3 + 3}{4 \cdot 2{,}3 - 3} \right) = 18{,}5 \text{ Ом} \tag{4.12}$$

5) Визначається теоретична кількість вертикальних заземлювачів n штук, без урахування коефіцієнта використання $\eta_в$:

$$n = \frac{2 \cdot R_в}{R_д} = \frac{2 \cdot 18{,}5}{4} = 9{,}25 \tag{4.13}$$

І визначається коефіцієнт використання вертикальних електродів групового заземлювача без врахування впливу з'єднувальної стрічки $\eta_в$ =0,57 (табличне значення).

6) Визначається необхідна кількість вертикальних заземлювачів з урахуванням коефіцієнта використання $n_в$, шт:

$$n_в = \frac{2 \cdot R_в}{R_д \cdot \eta_в} = \frac{2 \cdot 18{,}5}{4 \cdot 0{,}57} = 16{,}2 \approx 16 \tag{4.14}$$

7) Визначається довжина з'єднувальної стрічки горизонтального заземлювача $l_с$, м:

$$l_c = 1{,}05 \cdot L_в \cdot (n_в - 1), \tag{4.15}$$

де $L_в$ – відстань між вертикальними заземлювачами, (прийняти за $L_в$ = 3м);

$n_в$ – необхідна кількість вертикальних заземлювачів.

$$l_c = 1{,}05 \cdot 3 \cdot (16 - 1) \approx 48\text{м} \tag{4.16}$$

8) Визначається опір розтіканню струму горизонтального заземлювача (з'єднувальної стрічки) $R_г$, Ом:

$$R_г = \frac{\rho_{розр.г}}{2 \cdot \pi \cdot l_c} \cdot ln \frac{2 \cdot {l_c}^2}{d_{см} \cdot h_г}, \quad (4.17)$$

де $d_{см}$ – еквівалентний діаметр смуги шириною b, $d_{см}$ =0,95b, b = 0,15 м;

$h_г$ – глибина закладання горизонтальних заземлювачів (0,5 м);

l_c - довжина з'єднувальної стрічки горизонтального заземлювача l_c, м

$$R_г = \frac{220}{2 \cdot \pi \cdot 48} \cdot ln \frac{2 \cdot 48^2}{0{,}95 \cdot 0{,}15 \cdot 0{,}5} = 8{,}1 \text{ Ом} \quad (4.18)$$

9) Визначається коефіцієнт використання горизонтального заземлювача η_c. відповідно до необхідної кількості вертикальних заземлювачів $n_в$.

Коефіцієнт використання з'єднувальної смуги η_c=0,3 (табличне значення).

10) Розраховується результуючий опір заземлювального електроду з урахуванням з'єднувальної смуги:

$$R_{заг} = \frac{R_в \cdot R_г}{R_в \cdot \eta_c + R_г \cdot n_в \cdot \eta_в} \leq R_д. \quad (4.19)$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{заг}$ <4 Ом, а саме:

$$R_{заг} = \frac{18{,}5 \cdot 8{,}1}{18{,}5 \cdot 0{,}3 + 8{,}1 \cdot 16 \cdot 0{,}57} = 1{,}9 \leq R_д \quad (4.20)$$

3) При виникненню пожеж при роботі на ПЕОМ від таких можливими джерел запалювання як:

– іскри і дуги коротких замикань;

– перегрів провідників, резисторів та інших радіодеталей ПЕОМ, від тривалої перевантаження та наявність перехідного опору;

– іскри при розмиканні і розмиканні ланцюгів;

– розряди статичної електрики;

– необережному поводженню з вогнем, а також вибухи газо-повітряних і паро-повітряних сумішей.

Важливу увагу слід звернути на пожежну безпеку підприємства в цілому і окремих його приміщень. В приміщеннях не повинно накопичуватися сміття, непотрібний папір, мотлох та ін. речі, які не використовуються у виробничому процесі. Наявний вільний аварійний вихід за межі приміщення в разі пожежі, бути передбачені вогнегасники. Вони повинні бути в робочому стані і перевірятися згідно з нормами. У приміщеннях повинна бути пожежна сигналізація, вогнегасник. У разі виникнення пожежі необхідно повідомити в найближчу пожежну частину, убезпечити інших працівників і по можливості прийняти кроки по запобіганню можливих наслідків та усуненню пожежі. [52]

4.9 Техногенне забруднення навколишнього середовища

Сьогодні нас оточує величезна кількість бруду. Починаючи від простого пилу і закінчуючи шкідливими хімічними відходами на виробництві, залишаються непоміченими такі типи забруднення, як шумове, теплове та електромагнітне. Останні здебільшого впливають на здоров'я людини під час перебування у квартирах жилих будинках та офісах.

Виокремлюють чотири типи забруднення – інгредієнтне, біоценотичне, параметричне та стаціально-деструктивне. Група забруднень, яка найбільш пронизана техногенним впливом – параметрична. У цій групі містяться п'ять типів забруднень: теплове, шумове, сміттєве, радіаційне, та електромагнітне. Розглянемо окремо кожне з забруднень [53].

4.9.1 Теплове забруднення

Тепло, що виділяється внаслідок обробки, виробництва, переробки чи використання деяких ресурсів завжди є небажаним і надмірним, за виключенням випадків, коли метою було обігрівання об'єктів (наприклад, взимку). Теплове забруднення можна розділити на забруднення водних ресурсів та атмосфери.

Основними джерелами такого типу забруднення вод є атомні й теплові електростанції. Теплове забруднення прибережних морських акваторій і поверхні водойм виникає в результаті скидання нагрітих стічних вод. Це призводить до наступних наслідків:

– підвищення температури води часто підсилює сприйнятливість організмів до дії токсичних речовин;

– температура може перевищити критичні значення для «стенотермних» стадій життєвого циклу водних організмів;

– висока температура сприяє заміні звичайної флори водоростей на менш бажані синьо-зелені водорості, що викликають «цвітіння» води;

– при підвищенні температури води тваринам потрібно більше кисню, оскільки в теплій воді його вміст знижений у зв'язку з меншою розчинністю.

При викидах з труб промислових підприємств для опалення будинків, вихлопних труб двигунів внутрішнього згоряння, лісових пожежах виділяються речовини, нагріті до більше ніж до 60°С. Середньорічна температура атмосферного повітря над великими містами та промисловими центрами на 6-7 градусів вище температури повітря прилеглих територій. Фахівці відзначають, що в останні 25 років середня температура тропосфери піднялася на 0,7 градусів Цельсія [54].

4.9.2 Шумове забруднення

Шуми відносяться до числа шкідливих забруднень атмосфери. Їх подразнююча дія для людини залежить від спектрального складу, інтенсивності 186 та тривалості впливу. Найбільше роздратування викликає шум в діапазоні частот 3000-5000 Гц.

Чим гучніше грає музика у навушниках, тим швидше вухо буде звикати до такого рівня гучності і поступово зовсім перестане сприймати тихі звуки. Шум згубно діє не тільки на слуховий апарат, але і на центральну нервову систему людини, роботу серця, служить причиною багатьох інших захворювань.

Щодо роботи в умовах підвищеного шуму, то вона загострює слух на високих частотах та викликає швидку стомлюваність. При різних інтенсивностях шуму виникають різні недуги. Так при 145-140 дБ виникають вібрації у м'яких тканинах носа і горла, у кістках черепа та зубах. При інтенсивності більше за 140 децибел, з'являються біль у вухах і голові, починає вібрувати грудна клітина, м'язи рук та ніг, сильна втома й дратівливість. При рівні шуму понад 160 децибел може відбутися розрив барабанних перетинок.

4.9.3 Сміттєве забруднення

Зі збільшенням кількості міст та промислових підприємств постійно збільшується кількість відходів, які вони виробляють. Промислові та побутові відходи створюють безліч проблем, таких як транспортування, зберігання, утилізація та ліквідація. Особливої уваги варто приділити різній сучасній техніці, що рано чи пізно відправляється на сміттєзвалище. У підземні води потрапляють шкідливі та отруйні речовини. Для запобігання негативного впливу

на оточуючу місцевість та її екосистему необхідно утилізувати комп'ютерну техніку через спеціалізовані приймальні пункти.

Викидаючи сміття, люди порушують один з основних екологічних законів – кругообіг речовин у природі. Адже, вилучаючи з природи чимало речовин, людина змінює їх до невпізнанності та повертає у природу у вигляді сміття, яке не розкладається на вихідні речовини природним шляхом [55].

4.9.4 Електромагнітне забруднення

Кожного року ми спостерігаємо збільшення різноманітної техніки, технічний прогрес все більш прискорюється. Купуючи телефони, комп'ютери, побутову техніку, планшети та інші гаджети, ми намагаємось зробити своє життя якомога комфортнішим та простішим.

Побічні ефекти такого прогресу починаєш відчувати лише з плином часу. Повітря в квартирі починає ставати дедалі сухішим, а вільного місця для розташування різноманітної техніки з кожним днем потрібно все більше. Інша справа, що сумлінні виробники такої техніки самі починають усвідомлювати усі негативні її прояви, і починають прагнути до менш шкідливої продукції.

Серед незліченної кількості техніки, яка нас зараз оточує, електромагнітне випромінювання стає дедалі більш сильним. Несприятливий вплив на організм людини мають електромагнітні випромінювання промислової частоти (50 герц) та частот радіохвильового діапазону. У помешканнях електромагнітні поля різної потужності створюють будь-які електронні прилади – радіоапаратура, телевізори, холодильники тощо. Відповідно це створює певну небезпеку. І річ у тім, що кожен наш внутрішній орган працює на своїй певній частоті, наприклад, серце – близько 700 герц, мозок у стані сну – 10 герц, бадьорості – 50 герц [56]. Якщо поруч знаходиться постійне джерело електромагнітного випромінювання, яке працює на аналогічній, чи кратній, частоті, то це може призвести до збільшення або зменшення нормальної частоти роботи органу. Наслідком цього може бути головний біль, порушення сну, перевтома, стенокардія. Найнебезпечніший час для опромінення, це коли людина, а особливо дитина, перебуває у стані глибокого сну.

Щоб уникнути та попередити усі можливі проблеми, слід регулярно провітрювати приміщення, де знаходиться техніка, протирати пил та розташувати неподалік живі рослини. Важливо також і вміння організовувати свій час, робити перерви та фізичні вправи під час довготривалої роботи за комп'ютером. Необхідно частіше виїжджати на природу, вживати здорову їжу.

4.10 Висновки

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом написаному в кваліфікаційній роботі, описано, які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника.

Приведені рекомендації щодо організації робочого місця, а також важливу інформацію щодо пожежної та електробезпеки. Була наведена схема, розміри приміщення та наведено значення температури, вологості й рухливості повітря, необхідна кількість і потужність ламп та інші параметри, значення яких впливає на умови праці робітника, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп’ютері.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Під час написання дипломної роботи виконано ряд поставлених задач:

– проаналізовано існуючі методи забезпечення відмовостійкості то обрано мультиверсійний метод;

– створено модель надійного базового програмного забезпечення;

– спроектовано і включено програмні засоби для забезпечення живучості при програмних, інформаційних і апаратних відмовах;

– побудовано модель забезпечення відмовостійкості шляхом впровадження мультиверсійності;

– реалізовано програмне забезпечення – диспетчер відмовостійкої керуючої системи.

Удосконалено модель диспетчера керуючого обчислювального процесу шляхом дослідження організації паралельних обчислювальних процесів у мультиядерних мікропроцесорах, що дозволяє підвищити загальну відмовостійкість і зменшити витрати при впровадження її у систему.

Так як найбільш імовірним джерелом систематичних відмов систем управління є приховані дефекти програмних засобів, даний спосіб дозволяє знизити ризики відмов від прихованих помилок, оскільки в разі проектних дефектів зменшується ймовірність одночасної і однотипної відмови різних версій ПЗ. При застосуванні даного методу також підвищується відмовостійкість ПЗ за рахунок реалізації принципу мажорірованія «2 із 3» и підвищується коефіцієнт готовності керуючої системи. У системах управління з високими вимогами до надійності розглянутий спосіб може застосовуватися разом з традиційними способами резервування апаратури, наприклад, в контролерах з трьома мікропроцесорами.

В результаті виконання дипломної роботи досягнуто її мету, котра є дослідження способів організації надійного відмовостійкого обчислювального процесу в керуючому промисловому контролері і практична реалізація мультіверсійного програмування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Єлісєєв В.В., Ларгін В.А., Пивоваров Г.Ю. Програмно-технічні комплекси АСУ ТП: Навч. посібник - К .: Видавничо-поліграфічний центр «Київський університет», 2003.
2. ДСТУ 2861-94 Надійність техніки. Аналіз надійності. Основні положення.
3. Василевський О. М. Нормування показників надійності технічних засобів: навчальний посібник / О. М. Василевський, В. О. Поджаренко. — Вінниця: ВНТУ, 2010. — 129 с.
4. УСЕ (Універсальний словник-енциклопедія)
5. ДСТУ 2938-94 Основні поняття. Терміни та визначення.
6. Якість програмного забезпечення глава 10 [Електронний ресурс]: ISO / IEC TR 19759: 2005 Режим доступу: https://www.iso.org/obp/ui/#iso:std:iso-iec:tr:19759:ed-2:v2:en. (Дата звернення: 01.10.2018)
7. Єфімова Т. І., Мудла Б. Г., Шалейко О. М. Відмовостійкість програмного забезпечення гарантоздатних комп'ютерних систем //Математические машины и системы. – 2009. – Т. 1. – №. 4.
8. Резервування URL: https://uk.wikipedia.org/wiki/Резервування_(техніка) (дата звертання 21.11.2018)
9. Авізеніс, А. Про реалізацію N-версії програмування програмного відмовостійкості під час виконання. / А. Авізеніс, Л. Чен. В Proc. IEEE COMPSAC 77, с. 149-155, 1977.].
10. Єфімова Т. І., Мудла Б. Г., Шалейко О. М. Відмовостійкість програмного забезпечення гарантоздатних комп'ютерних систем //Математические машины и системы. – 2009. – Т. 1. – №. 4.
11. Р. Т. Вуд Різноманітні стратегії, спрямовані на пом'якшення наслідків вразливості, що виникла внаслідок загальних причин. Сьоме американське ядерне товариство Міжнародне місце зустрічі з ядерних установок інструменти, контрольні та людських-машинних інтерфейсних технологій NPIC & HMIT 2010, Лас-Вегас, штат Невада, 7-11 листопада 2010 року на CD-ROM, Американське ядерне товариство, LaGrange Park, IL (2010).
12. IEC 61508: 1-6: 2012. Функціональна безпека електричних / електронних і програмованих електронних систем. Частина 3. Вимоги до програмного забезпечення.
13. Гнеденко Б.В., Бєляєв Ю.К., Соловйов А.Д., Каштанов В.А. Математичні методи в теорії надійності. 2013. Книжковий дім «ЛІБРОКОМ». М. 550 с.
14. William Stallings. Operating Systems - Internals and Design Principles, 7th Edition. — Prentice Hall, 2011. — 816 p. — ISBN 013230998X. (англ.)

15. Програмувальні контролери - Частина 3: Мови програмування [Електронний ресурс]: IEC 61131-3:2013 Режим доступу: https://webstore.iec.ch/publication/4552. (Дата звернення: 04.10.2018)

16. Харченко В. С., Скляр В. В., Токарев В. І. Моделі відмовобезпечних структур цифрових систем контролю і управління // Системи ОБРОБКИ інформації.-Х .: ХВУ.-2003 Вип. - 2003. - Т. 4. - С. 200-205.

17. Харченко В.С. Інформаційна технологія підтримки визначення компонентних функціональних структур живучих бортових інформаційно-керуючих систем / В.С. Харченко, Н.П. Бородавка // Збірник наукових праць Харківського національного університету Повітряних Сил. - 2007. - № 1 (13). - С. 82-86.

18. Харченко В.С. Автоматні моделі багатоверсійних інформаційно-керуючих систем / В.С. Харченко, В.В. Скляр, В.А. Головір // Системи ОБРОБКИ інформації. - 2007. - № 1 (59). - С. 108-109.

19. М.А. Ястребенецький, В.Н. Васильченко, С.В. Виноградська та ін. Безпека атомних станцій: Інформаційні управляючі системи: монографія /; під ред. М.Я. Ястребенецький. К.: Техніка, 2004. 472 с.

20. В. В. Скляр, В. С. Харченко, "Відмовостійкі комп'ютерні системи управління з версійно-порогової адаптацією: способи адаптації, оцінка надійності, вибір архітектури", Автомат. і телемех., 2002 № 6, 131-145; Autom. Remote Control, 63: 6 (2002), 991-1003

21. Волкова А.В. та ін. Масштабуються багатоверсійності технології для критичних додатків. Лекції та практикум / Харченко В.С. (Редактор). - МОН України, ХАІ, 2013. - 202с.

22. Соколов Ю.М. Інструментальне оцінювання надійності програмно-технічних комплексів при зростанні інтенсивності відмов / Ю.М. Соколов, В.С. Харченко, Ю.Л. Поночовний // Системи ОБРОБКИ інформації. - 2014. - № 2 (118). - С. 205-211.

23. Брежнєв Є.В. Метод оцінювання безпеки критичної енергетичної інфраструктури з урахуванням надійності цифрової підстанції / Є.В. Брежнєв, В.С. Харченко // Наука і техніка Повітряних Сил Збройних Сил України. - 2014. - № 3 (16). - С. 138-143.

24. А.В. Федухін, А.А. Муха. Забезпечення живучості систем протиаварійної автоматики на гідроелектростанціях. Математичні машини і системи, 2018, № 2. Стор. 169- 194.

25. Якимець Н.В. Відмовостійкі цифрові системи керування з програмованою логікою на основі частково працездатних автоматів: моделі та реалізація / Н.В. Якимець, В.С. Харченко // Системи ОБРОБКИ інформації. - 2007. - № 4 (62). - С. 134-138.

26. С. Золотарев. Операционные системы реального времени для 32-разрядных микропроцессоров

27. The Open Group The Single UNIX Specification, version 2

28. What makes a good RTOS, Dedicated Systems Experts NV, June 11, 2001

29. И. Б. Бурдонов, А. С. Косачев, В. Н. Пономаренко. Операционные системы реального времени п. 1. Введение: Особенности операционных систем реального времени

30. А. А. Жданов. Операционные системы реального времени

31. Е. Хухлаев. Операционные системы реального времени и Windows NT

32. D. Kalinsky. Basic concepts of real-time operating systems. Архів оригіналу за 2013-01-28.(англ.)

33. А. А. Блискавицкий, С. В. Кабаев. Операционные системы реального времени (обзор)

34. И. Б. Бурдонов, А. С. Косачев, В. Н. Пономаренко. Операционные системы реального времени п. 1.2. Планирование, приоритеты

35. NUMA https://uk.wikipedia.org/wiki/NUMA (дата звертання: 20.11.18)

36. Spracklen L., Abraham S. G. Chip multithreading: Opportunities and challenges //High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on. – IEEE, 2005. – С. 248-252.

37. Marr D. et al. Hyper-threading technology in the netburst® microarchitecture //14th Hot Chips. – 2002.

38. Todd Brian, Putting multicore processing in context: Part one. Embedded.com, February 2006.

39. Стахів П. Г., Струбицька І. П. Метод розпаралелення задачі ідентифікації параметрів дискретних динамічних макромоделей на масивно-паралельних процессорах //Наукові нотатки. – 2010. – №. 27. – С. 300-305.

40. Владова А. Ю. Разработка масштабируемых программ для многоядерных архитектур: лабораторный практикум. – 2006.

41. Barker B. Message passing interface (mpi) //Workshop: High Performance Computing on Stampede. – 2015. – Т. 262.

42. Huang L. et al. Parallel Processing Transport Model MT3DMS by Using OpenMP //International Journal of Environmental Research and Public Health. – 2018. – Т. 15. – №. 6. – С. 1063.

43. Silberschatz A., Gagne G., Galvin P. B. Operating system concepts. – Wiley, 2018.

44. Рольщиков В. Б. Технології розподілених систем та паралельниї обчислень. Конспект лекцій (Змістовний модуль No1). – 2018.

45. Spitzer C. R. Avionics: elements, software and functions. – CRC Press, 2016.

46. Alvioli M., Baum R. L. Parallelization of the TRIGRS model for rainfall-induced landslides using the message passing interface //Environmental Modelling & Software. – 2016. – Т. 81. – С. 122-135.

47. Message_Passing_Interface https://uk.wikipedia.org/wiki/Message_Passing_Interface (дата звертання: 29.11.18)

48. НПАОП 0.00-1.28-10 Правила охорони праці під час експлуатації електронно-обчислювальних машин

49. ДСанПіН 3.3.2.007-98 Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин

50. ДСН 3.3.6.042-99 Санітарні норми мікроклімату виробничих приміщень.

51. ДБН В.2.5-28:2015 Освітлення у примішеннях

52. НПАОП 40.1-1.01-97 Класифікація приміщень за ступенем небезпеки ураження електричним струмом.

53. Н. А. Агаджанян, В.І. Торшин. «Екологія людини». ММП «Екоцентр», КРУК, 1994.

54. В. М. Лапін. «Безпека життєдіяльності людини». – Київ – Львів, 1999.

55. Є. О. Кріксунов, В.В. Пасічник, А.П. Сидорин. «Екологія». Видавничий дім «Дрофа», 1995.

56. Є. П. Желібо, Н. М. Заверуха, В. В. Зацарний «Безпека життєдіяльності». Видавництво «Каравела», Київ, 2003.

Додаток А

Код програми

```
#define maxHandlers 64
#define maxMessages 64
#define maxTimers  64

#define MSG_ADC_CYCLE    1
#define MSG_KEY_PRESS    2
#define MSG_KEY_REPEAT    3
#define MSG_LCD_REFRESH    4
#define MSG_BRESENHAM    5
#define MSG_MENU_SELECT    6
#define MSG_MENU_CANCEL    7
#define MSG_1W_TEMP      8
#define MSG_CHECK_TEMP    9
#define MSG_DISP_REFRESH  10
#define MSG_TIMER_SEC    11

typedef unsigned char msg_num;
typedef int msg_par;
typedef unsigned char(*handler)(msg_par);


typedef struct {
    msg_num msg;
    handler hnd;
} iHandler;


typedef struct {
    msg_num msg;
    msg_par par;
} iMessage;


typedef struct {
    msg_num msg;
    msg_par par;
    unsigned int time;
} iTimer;

iTimer    lTimer[maxTimers];
iHandler  lHandler[maxHandlers];

iMessage  lMessage[maxMessages];
unsigned int lMesPointer = 0, hMesPointer = 0;

void setHandler(msg_num msg, handler hnd) {
    unsigned char i, j;
    i = 0; j = 0;
    while (i < maxHandlers) {
        if (lHandler[i].msg == 0) {
            lHandler[i].hnd = hnd;
            lHandler[i].msg = msg;
            break;
        }
        i++;
    }
}


void killHandler(msg_num msg, handler hnd) {
    unsigned char i, j;
    i = 0; j = 0;
    while (i < maxHandlers) {

        if ((lHandler[i].msg == msg) && (lHandler[i].hnd == hnd)) {
            lHandler[i].msg = 0;
        }

        if (lHandler[i].msg != 0) {
            if (i != j) {
```

```
                lHandler[j].msg = lHandler[i].msg;
                lHandler[j].hnd = lHandler[i].hnd;
                lHandler[i].msg = 0;
            }
            j++;
        }
        i++;
    }
}

void sendMessage(msg_num msg, msg_par par) {
    hMesPointer = (hMesPointer + 1) & (maxMessages - 1);

    lMessage[hMesPointer].msg = msg;
    lMessage[hMesPointer].par = par;
    if (hMesPointer == lMesPointer) {
        lMesPointer = (lMesPointer + 1) & (maxMessages - 1);
    }
};


void dispatchMessage() {
    char i;
    unsigned char res;

    if (hMesPointer == lMesPointer) {
        return;
    }

        lMesPointer = (lMesPointer + 1) & (maxMessages - 1);

        msg_num msg = lMessage[lMesPointer].msg;
        msg_par par = lMessage[lMesPointer].par;

        if (msg == 0)
            return;

        for (i = maxHandlers - 1; i >= 0; i--) {
            if (lHandler[i].msg == msg) {
                res = lHandler[i].hnd(par);
                if (res) {
                    break;
                }
            }
        }
    }

    void setTimer(msg_num msg, msg_par par, unsigned int time) {
        unsigned char i, firstFree;
        firstFree = maxTimers + 1;
        if (time == 0) {
            sendMessage(msg, par);
        }
        else {

            for (i = 0; i < maxTimers; i++) {
                if (lTimer[i].msg == 0) {
                    firstFree = i;
                }
                else {
                    if ((lTimer[i].msg == msg) && (lTimer[i].par == par)) {
                        lTimer[i].time = time;
                        return;
                    }
                }
            }
            if (firstFree <= maxTimers) {
                lTimer[firstFree].msg = msg;
                lTimer[firstFree].par = par;
                lTimer[firstFree].time = time;
```

```
141.                 }
142.             }
143.         }
144.
145.         void killTimer(msg_num msg) {
146.             unsigned char i;
147.             for (i = 0; i < maxTimers; i++) {
148.                 if (lTimer[i].msg == msg) {
149.                     lTimer[i].msg = 0;
150.                 }
151.             }
152.         }
153.
154.         void dispatchTimer() {
155.             unsigned char i;
156.             msg_num msg;
157.             msg_par par;
158.
159.             for (i = 0; i < maxTimers; i++) {
160.                 if (lTimer[i].msg == 0)
161.                     continue;
162.
163.                 if (lTimer[i].time == 0) {
164.                     msg = lTimer[i].msg;
165.                     par = lTimer[i].par;
166.                     lTimer[i].msg = 0;
167.                     sendMessage(msg, par);
168.                 }
169.                 else {
170.                     lTimer[i].time--;
171.                 }
172.             }
173.         }
```

Додаток Б

Презентація

ДОСЛІДЖЕННЯ МЕТОДІВ ПІДВИЩЕННЯ ВІДМОВОСТІЙКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПРОМИСЛОВОГО КОНТРОЛЕРА

КЕРІВНИК ДИПЛОМНОЇ РОБОТИ: К.Т.Н. ДОЦЕНТ ЛАРГІН ВІКТОР АНАТОЛІЙОВИЧ

СТУДЕНТ: БЕРЕЖНИЙ АНАТОЛІЙ ГЕННАДІЙОВИЧ

АКТУАЛЬНІСТЬ ТЕМИ І МЕТА ПРОЕКТУ

- Однією з основних проблем при розробці систем управління критичними об'єктами є створення високонадійного програмного забезпечення керуючих контролерів.
- Найбільш імовірним джерелом систематичних відмов систем управління є дефекти програмних засобів, внесені при розробці, що не виявлені при тестуванні і верифікації і які з'являються при певному наборі вхідних даних або характеристиках фізичних або інформаційних середовищах.
- **Метою даної роботи є** дослідження способів організації надійного відмовостійкого обчислювального процесу в керуючому промисловому контролері і практична реалізація мультиверсійного програмування.

Постановка задачі

- АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ ВІДМОВОСТІЙКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
- ПОБУДОВА МОДЕЛІ ЗАБЕЗПЕЧЕННЯ ВІДМОВОСТІЙКОСТІ ШЛЯХОМ ВПРОВАДЖЕННЯ МУЛЬТИВЕРСІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
- РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ – ДИСПЕТЧЕРА ВІДМОВОСТІЙКОЇ КЕРУЮЧОЇ СИСТЕМИ

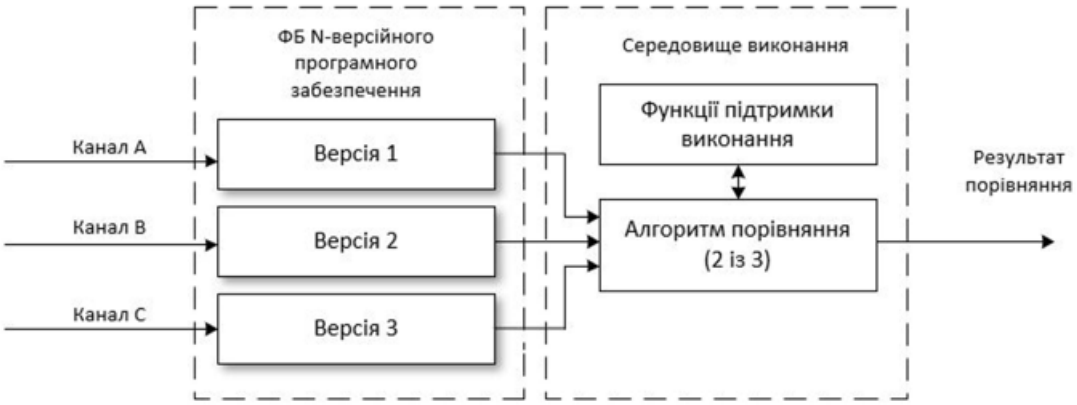
ВИДИ НАДМІРНОСТІ

В даний час в обчислювальних системах широке поширення отримав цілий ряд методів забезпечення відмовостійкості, розглянутих в ряді робіт провідних вчених.

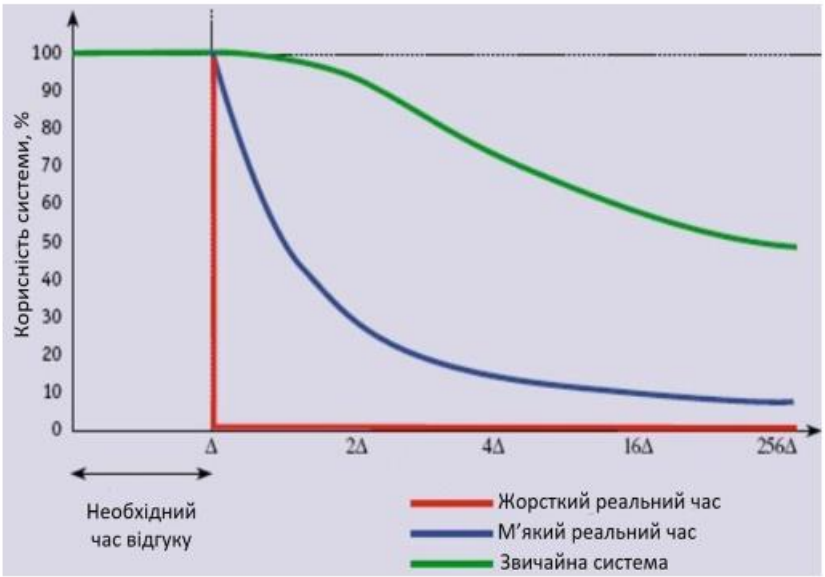
Незважаючи на їх різноманіття, дані методи можна розділити на три основні групи, які в своїй основі використовують такі види надмірності:

- апаратну надмірність;
- часова надмірність;
- програмна надмірність.

МУЛЬТИВЕРСІЙНІСТЬ



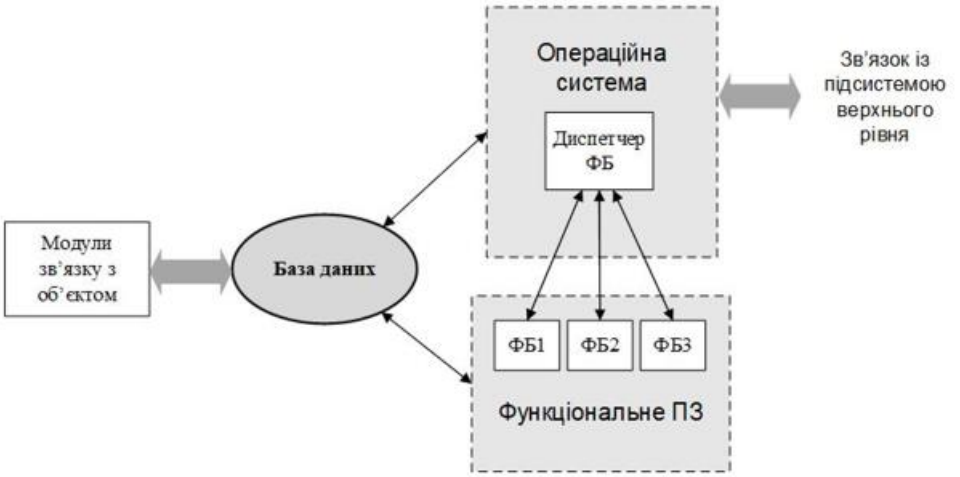
5



ОПЕРАЦІЙНА СИСТЕМА РЕАЛЬНОГО ЧАСУ

6

ОРГАНІЗАЦІЯ ТРЬОХВЕРСІЙНОГО ПРОЦЕСУ



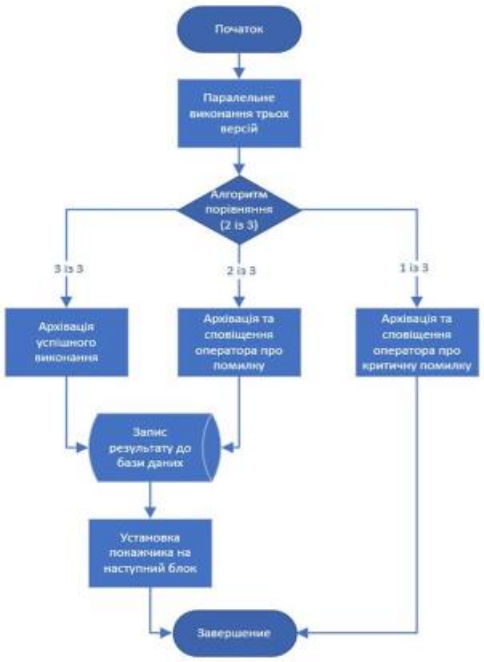
7

ЦИКЛІЧНЕ ВИКОНАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Алгоритм виконання циклу ПЗ наступний:

- запуск трьох модулів ФБ в мікроконтролері;
- виконання модулів ФБ;
- формування результату по принципу мажорірованія «2 із 3»;
- сповіщення оператора, якщо на одному з ланок знайдено помилку;
- запис результату в БД;
- установ покажчика в списку викликів на наступний ФБ.

8



БЛОК-СХЕМА ФОРМУВАННЯ РЕЗУЛЬТАТУ ПОРІВНЯННЯ ПО МАЖОРИТАРНОМУ ПРИНЦИПУ

ВИСНОВКИ

В результаті виконання магістерської роботи отримано такі результати:

- проаналізовано існуючі методи забезпечення відмовостійкості та обрано мультиверсійний метод;
- побудовано модель забезпечення відмовостійкості шляхом впровадження мультиверсійності;
- реалізовано програмне забезпечення – диспетчер відмовостійкої керуючої системи.

ПЕРСПЕКТИВИ РОЗВИТКУ

На сьогодні у підприємстві СНПО «Импульс» ведеться розробка нового промислового контролера МСКУ-5 на базі багатоядерного мікропроцесора.

У системах управління на АЕС зазвичай використовують три окремі мікроконтролери.

Якщо використовувати усі чотири ядра кожного мікроконтролера, це збільшить відмовостійкість системи. Головним нововведенням є трьохверсійне відмовостійке програмування на базі новітніх чотирьохядерних мікроконтролерів Intel із кожною версією на одне з ядер. Тому, впровадження мультиверсійності в новітніх мікроконтролерах є запорукою стабільного технологічного процесу.

ДЯКУЮ ЗА УВАГУ