МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Кафедра Комп'ютерні інформаційні технології

«ДО ЗАХИСТУ»

Завідувач кафедри
_____ /В. І. Шинкаренко/

«_____» _____ 20____р.

ДИПЛОМНА РОБОТА
на здобуття освітнього ступеня «Магістр»

Галузь знань **12 Інформаційні технології**

Спеціальність **121 Інженерія програмного забезпечення**

Тема **Дослідження ефективності засобів синхронізації потоків в ОС QNX**

Theme **Research of efficiency threads synchronization primitives in OS QNX**

Керівник дипломної роботи доц. _____ В. Я. Нечай

Нормоконтролер доц. _____ О. С. Куроп'ятник

Студент групи ПЗ1921 _____ Д. С. Сєнін

Student Sienin Dmytro

Дніпро
2020

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Факультет Комп'ютерних технологій і систем Кафедра Комп'ютерні інформаційні технології

Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
_____ проф. Шинкаренко В.І.
(підпис)
«___»_____2020 р.

ЗАВДАННЯ

до дипломної роботи на здобуття ОС Магістр
(освітній ступень)

студента групи (ПЗ1921) 961-М Сєніна Дмитра Святославовича
(номер групи) (ПІБ)

1 Тема дипломної роботи: Дослідження ефективності засобів синхронізації потоків в ОС QNX затверджена наказом по університету від «10» жовтня 2019 р. № 779ст.

2 Термін подання студентом закінченого проекту «16» грудня 2020 р.

3 Вихідні дані до дипломного проекту
_____
_____
_____

4 Зміст пояснювальної записки (перелік питань до розробки) Призначення, постановка задачі та огляд аналогів, методика проведення дослідження, проектування і розробка інструментального програмного забезпечення, дослідження ефективності засобів синхронізації, охорона праці та безпека в надзвичайних ситуаціях, висновки.

5 Перелік демонстраційного матеріалу Постановка задачі, опис аналогів, методи вирішення, механізми реалізації, аналіз результатів, висновки.

6. Консультанти (з назвами розділів):

Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Техніко-економічні розрахунки	доц. Гненний М.В.		
Охорона праці та безпека в надзвичайних ситуаціях	ст. викладач Музикін М.І.		

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва розділів дипломного проекту	Термін виконання розділів проекту (роботи)	Примітка
1	Вступ	1.09.2020 – 10.09.2020	
2	Огляд літератури	10.09.2020 – 15.09.2020	
3	Постановка задачі, технічне завдання	15.09.2020 – 30.09.2020	
4	Створення тестової програми	01.10.2020 – 15.10.2020	
5	Перші тестування	15.10.2020 – 22.10.2020	30%
6	Аналіз результатів	30.10.2020 – 11.11.2020	
7	Розрахунок економічних показників	11.11.2020 – 14.11.2020	
8	Охорона праці	14.11.2020 – 18.11.2020	60%
9	Оформлення пояснювальної записки	18.11.2020 – 01.12.2020	
10	Демонстраційні матеріали	5.11.2020 – 16.12.2020	100%

Дата видачі завдання «10» жовтня 2019 р.

Керівник дипломного проекту _____ Нечай В.Я.
(підпис) (ПІБ)

Завдання прийняв до виконання _____ Сєнін Д .С.
(підпис) (ПІБ)

РЕФЕРАТ

Об’єктом даного дослідження є статистична обробка отриманих часових характеристик роботи примітивів синхронізації в операційній системі QNX.

Предметом дослідження є вплив часу виконання задач на часові характеристики засобів синхронізації в ОСРЧ QNX.

Метою даної роботи є порівняння часу виконання різноманітних примітивів синхронізації в операційній системі реального часу QNX на різних типах задач з метою визначення який з примітивів є найбільш ефективний в кожному з протестованих випадків.

Результати та їх новизна: доцільність використання того чи іншого примітиву синхронізації в залежності від складності задачі загалом є дослідженою, проте не в операційних системах реального часу. Дане дослідження робить свій внесок розробку рекомендацій щодо застосування примітивів синхронізації в протестованих типах задач.

Дана пояснювальна записка складається з 8 розділів:

1. вступ – визначає актуальність роботи, її цілі і головні задачі. Складається з 3 сторінок;
2. обгрунтування методу дослідження часової ефективності засобів синхронізації потоків – описує основні аспекти вибору методу дослідження. Складається з 10 сторінок;
3. проектування та розробка інструментального забезпечення – містить інженерно-технологічну постановку задачі, вибір мови програмування і середовища розробки, об’єктно-орієнтоване проектування. Складається з 39 сторінок;
4. аналіз результатів – міститить результати проведених досліджень. Складається з 24 сторінок;

5. охорона праці та безпека у надзвичайних ситуаціях – містить аналіз шкідливих та небезпечних факторів при роботі з ЕОМ, та заходи щодо їх мінімізації. Складається з 9 сторінок;
6. аналіз результатів та висновки – складається з 2 сторінок;
7. бібліографічний список – список використаних літературних джерел. Складається з 9 сторінок.

Технічне завдання і робочий проект знаходяться в додатках.

Кількість таблиць: 29.

Кількість рисунків: 34.

Ключові слова: операційні системи реального часу (RTOS), задачі реального часу, примітиви синхронізації, статистичний аналіз.

ЗМІСТ

ВСТУП

Передові досягнення науки і техніки в області автоматичного керування процесами, такі як безпілотний автомобіль або система “розумного дому” загалом є дуже критичними до часу виконання завдань, які на них покладаються. Задачі, які вирішують більшість автономних систем є такими:

- задачі жорсткого реального часу (коли перевищення часу виконання поставлених завдань може призвести до невідворотних наслідків);
- задачі м’якого реального часу (коли перевищення часу вирішення небажане, але припустиме);
- задачі некритичні до часу виконання.

Актуальність роботи. Операційні системи та званого «широкого вжитку» такі як, наприклад, операційна система Windows нездатна виконувати задачі реального часу через те що мусить виконувати безліч супутніх функцій для спрощення роботи користувача з ОС. Саме тому ми звертаємо нашу увагу на операційні системи реального часу (ОСЧР), а саме на ОСРЧ QNX, яка є однією з найкращих концепцій мікроядерних операційних систем.

В процесі проектування та розробки програмного забезпечення під ОСРЧ де існує декілька потоків та процесів необхідно використовувати примітиви синхронізацій для правильної організації роботи з ними. Вибір правильного примітиву синхронізації є досить нетривіальною задачею бо залежить від багатьох чинників (складності задачі, типу задачі та ін). Також правильний вибір примітивів синхронізації може значно зменшити часові характеристики виконання програм. Дослідження проведені в цій області не дають в повній мірі відповіді на питання де і коли краще використовувати той чи інший примітив синхронізації. В зв’язку з цим було прийнято рішення протестувати деякі примітиви синхронізації на тривіальних задачах щоб визначити їх вклад в загальний час виконання програм і на основі цих даних розробити певні рекомендації щодо використання того чи іншого засобу синхронізації.

В даній роботі для синхронізації процесів та потоків в ОСРЧ QNX були протестовані наступні засоби:

- іменований семафор;
- неіменований семафор;
- м’ютекс;
- умовна змінна;
- сліпон.

Мета. Метою даної роботи є порівняння часу виконання різноманітних примітивів синхронізації в операційній системі реального часу QNX на різних типах задач з метою визначення який з примітивів є найбільш ефективний в кожному з протестованих випадків.

Об’єкт і предмет дослідження. Об’єктом даного дослідження є статистична обробка отриманих часових характеристик роботи примітивів синхронізації в операційній системі QNX. Предметом дослідження є вплив часу виконання задач на часові характеристики засобів синхронізації в ОСРЧ QNX.

Завдання. В процесі розробки програми і написання документації необхідно було вирішити наступне:

- розробити програмний продукт, що отримує часові характеристики виконання всієї програми та час виконання протестованого примітиву синхронізації для кожного з випадків;
- перенести отримані дані в MS Excel 2016 де здійснити статистичну обробку даних та їх візуалізацію для кращого сприйняття отриманої інформації;
- проаналізувати отримані результати та на їх основі розробити рекомендації щодо доцільності використання примітиву синхронізації в кожному з випадків.

Методика. Порівняння часових характеристик виконання примітивів синхронізації для кожного з випадків на отриманій тестовій вибірці.

Наукова новизна. Доцільність використання того чи іншого примітиву синхронізації в залежності від складності задачі загалом є дослідженою, проте не в операційних системах реального часу. Дане дослідження робить свій внесок розробку рекомендацій щодо застосування примітивів синхронізації в протестованих типах задач.

Практична значимість. Результати проведеного дослідження дозволяють на основі отриманих результатів розробити рекомендації щодо використання протестованих примітивів синхронізації в певних типах задач які можуть бути реалізовані в ОСРЧ QNX.

Апробація результатів дослідження. Процес та результати дослідницької роботи доповідались на семінарі кафедри КІТ 09.10.2020р, а також було опубліковано тези доповідей до щорічної міжнародної науково-практичній конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті».

Публікації за темою роботи. Підготовлено тези «Оцінка впливу випадкових характеристик обчислювальних процесів на найгірший час виконання програм (Worst Case Execution Time, WCET)», що були опубліковані за результатами проведення 13 Міжнародної науково-практичної конференції «Сучасні інформаційні і комунікаційні технології на транспорті, в промисловості та освіті».

1 ОГЛЯД ПРОБЛЕМИ

1.1 Призначення та область застосування

Сучасне програмне забезпечення, зокрема багатофункціональні системи, такі як автопілот або система автоматичного керування виробничою лінією підприємства є досить критичними до часу виконання задач які вони виконують. Також з цього виходить що ці системи є багатопотоковими або навіть багатопроцесорними і тому справедливо виникає питання синхронізації потоків та процесів. Від правильності обрання примітиву синхронізації не в останню чергу залежить і загальний час виконання програми. Було проведено багато досліджень на цю тему, проте проведені дослідження з ефективності засобів синхронізації в операційних системах реального часу, зокрема в ОС QNX є не достатньо систематизованими.

Програмна система «QNX C++ SyncPrimComp» реалізує отримання часових характеристик реалізованих примітивів синхронізації та загального часу виконання програми. Обрані примітиви синхронізації для порівняння умовно розбиваються на дві групи:

- умовна зміна та сліпон;
- семафор (іменований та неіменований) та м’ютекс.

В свою чергу всі ці примітиви порівнюються між собою на трьох основних задачах:

- двопотокова задача спільним ресурсом якої є зміна яку необхідно збільшити;
- двопотокова задача спільним ресурсом якої є цикл на 100000 ітерацій;
- задача з реалізацією трьох потоків спільним ресурсом якої є цикл на 100000 ітерацій.

Отримані часові характеристики оброблюються потім в середовищі Excel 2016 для визначення основних статистичних характеристик (дисперсії, відсоткового співвідношення виконання примітиву синхронізації до загального часу роботи програми) та побудови графіків, які наглядно демонструють результат ефективності роботи примітивів синхронізації в кожній з груп.

Підставою виникнення необхідності для розробки даного продукту стала відсутність прямих аналогів у відкритому доступі, а також складність модифікації та використання вже реалізованих програмних рішень.

Функціональне призначення даного програмного продукту полягає в отримані часових характеристик роботи примітивів синхронізації і загального часу виконання програм, а також статистична обробка даних (медіана, максимум, мінімум, розмах варіації, дисперсія, середньоквадратичне відхилення, коефіцієнт варіації та осциляції, процентне співвідношення виконання примітиву синхронізації до загального часу виконання програми).

Експлуатаційне призначення програми полягає в наданні програмісту інструменту для визначення оптимального з точки зору часових характеристик примітиву синхронізації для змодельованих тестових задач.

Областю застосування програмного продукту може слугувати як сфера програмування в цілому, так і сфера навчання. В сфері програмування, він може використовуватись для визначення оптимального примітиву синхронізації та/або розробки на основі цих даних гібридного примітиву, а в сфері навчання програма може використовуватись викладачами і студентами, наприклад, для порівняння часу роботи примітиву синхронізації двох подібних програм.

1.2 Постановка задачі

Дослідження рівня впливу програмного середовища операційних систем реального часу і отримання часових характеристик виконання примітиву синхронізації та програми для їх подальшої статистичної обробки і візуалізації в програмі Excel 2016.

Отримання часових характеристик передбачає розробку власного програмного продукту, який реалізує поставлені задачі. Тому необхідно розробити програмний продукт «QNX C++ SyncPrimComp», що дозволяє визначити час виконання програм та примітивів синхронізації на С і потім використати отримані результати для статистичного аналізу та візуалізації.

Основний функціонал програми повинен полягати в наступному: користувач запускає змодельовані тестові задачі що розроблені для кожного з примітивів синхронізації (по три для кожного з обраних примітивів) отримує часові характеристики програм, які потім використовує в програмі Excel 2016 (переносить у відповідні поля) для статистичної обробки та візуалізації результатів.

Програмний продукт повинен надавати програмісту наступні можливості:

- введення і редагування вихідного коду;
- збереження в файл коду;
- можливість компілювати і запускати вихідний код, при наявності відповідного компілятора;
- мінімальне налаштування інтерфейсу який надає термінал операційної системи;
- вивід в термінал отриманих часових характеристик роботи примітиву синхронізації та загального часу виконання програми;
- збереження в текстовий файл отриманих часових характеристик.

Програма повинна включати:

- інтерфейсну частину (головне вікно, яке надає термінал операційної системи);

- функціональну частину (файли розроблені на С для отримання часових характеристик);

- допоміжну частину (статистичний та графічний інтерпретатор реалізований в Excel 2016).

1.3 Огляд літературних джерел

Тема дослідження та оцінки часу виконання програм та примітивів синхронізації є і завжди була актуальною, оскільки будь-які системи критичні до часу реакції потребували часової верифікації. На сьогоднішній день розроблено багато різних методів отримання часової оцінки програм, кожен з яких має свої переваги і недоліки. Основними джерелами для ознайомлення з цими методами стала робота I. Ungurean «Timing Comparison of the Real-Time Operating Systems for Small Microcontrollers» [1], книга Цірюлік О., Горошко Е. «QNX/UNIX Анатомія паралелізму» [2] та електроний конспект лекції P. Puschener «Time analysis of security-critical real-time systems» Віденського технологічного університету [3]. Вони надають огляд основних методів оцінки виконання програм і основних методів їх програмної реалізації. Більш детальна інформація про ці методи була отримана з багаточисленних наукових робіт і статей (можна виділити таких авторів, як Reinhard Wilhelm [4], Олег Нікольский [5], Andreas Ermedahl [6] та Дмитро Волошин [7]), а також електронних конспектів вітчизняних і зарубіжних вищих навчальних закладів.

Інформацію в якій наведено деякі результати продуктивності операційної системи реального часу QNX Neutrino наведено в роботі Володимира Махільова «Результаты тестов производительности QNX Neutrino» [8].

Базову інформацію про примітиви синхронізації, а також про потоки та процеси було отримано з офіційної документації операційної системи реального часу QNX, що розмішена на головному сайті [9]. Також доречною стала книга автора D.R. Butenhof «Programing with POSIX Threads» [10].

Інформацію щодо статистичної обробки та візуалізації часових характеристик виконання примітивів синхронізації та загального часу виконання

програм було отримано з роботи Е. Чекотовського «Статистические методы на основе Microsoft Excel 2013» [11].

1.4 Огляд показників статистичної обробки часу

Медіаною називають таке значення ознаки, яке поділяє ранжирований ряд розподілу на дві рівні частини, тобто значення, яке перебуває в середині ряду розподілу [12]. Якщо в дискретному варіаційному ряду 2т + 1 випадків, то значення ознаки у випадку $m + 1$ є медіанним. Якщо в ряду парне число 2**т** випадків, медіану визначають як середню арифметичну з двох середенних значень. Дане значення дозволяє нам отримати середнє значення роботи примітиву синхронізації на одному з тестованих потоків. Воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

Максимум з тестової вибірки даних та генеральної сукупності це таке з чисел яке не менше чим всі інші числа [13]. Воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

Мінімум з тестової вибірки даних та генеральної сукупності це таке з чисел яке не більше чим всі інші числа [14]. Воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

Розмах варіації представляє собою різницю між максимальним і мінімальним значеннями ознаки [15]. В інтервальних рядах розподілу розмах варіації визначають як різницю між верхньою межею останнього та нижньою межею першого або як різницю між серединами інтервалів.

Безумовною перевагою показника розмаху варіації є простота його розрахунку. Однак він не може в повній мірі охарактеризувати варіацію ознаки, оскільки не враховує всіх значень ознаки, проміжних між максимальним та мінімальним значеннями. Не враховує він і частот. Особливість показника

розмаху варіації полягає у тому, що він залежить лише від двох крайніх значень ознаки, які можуть виявитися не достатньо типовими. В зв'язку з цим розмах варіації відображає інколи випадкове, а не типове для даного ряду коливання. Зазначені недоліки розмаху варіації звужують область його практичного застосування. В основному він використовується для попередньої оцінки варіації. Тому необхідні інші показники варіації, які ґрунтуються на всіх значеннях ознаки в даній сукупності. В даному випадку воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

Середнє лінійне відхилення являє собою арифметичну з абсолютних значень усіх відхилень індивідуальних значень ознаки від середньої [16]:

а) просте: $l = \sum \left| x - x_{сер} \right| / n$

б) зважене: $l = \sum \left| x - x_{сер} \right| f / \sum f$

Наявність абсолютних значень відхилень від середньої пояснюється так: середня арифметична має нульову властивість, згідно якої сума відхилень індивідуальних значень ознаки зі своїми знаками дорівнює нулю; щоб мати суму всіх відхилень, відмінних від нуля, кожне з них слід брати за абсолютною величиною.

Основним недоліком середнього лінійного відхилення є те, що в ньому не враховуються знаки відхилень, тобто їх спрямованість. Тому цей показник варіації використовується рідко (аналіз складу працюючих, ритмічність виробництва, обертання коштів у зовнішній торгівлі тощо). Прагнення мати показник варіації, який би усунув недоліки середнього лінійного відхилення, є дисперсія та лінійне квадратичне відхилення. В даному випадку воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

Дисперсія — міра розкиду даної отриманої величини, тобто її відхилення від середнього арифметичного значення [17]. Позначається *DX* у вітчизняній літературі й *var* (англ. *variance*) у зарубіжній. У статистиці часто вживається позначення σ^2_x або σ^2.

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N}$$

Тут x_i – значення отриманої величини; x – середнє значення; N - обсяг (кількість випадкових величин) генеральної сукупності.

Графік дисперсії випадкової величини наведено на рисунку 1.1

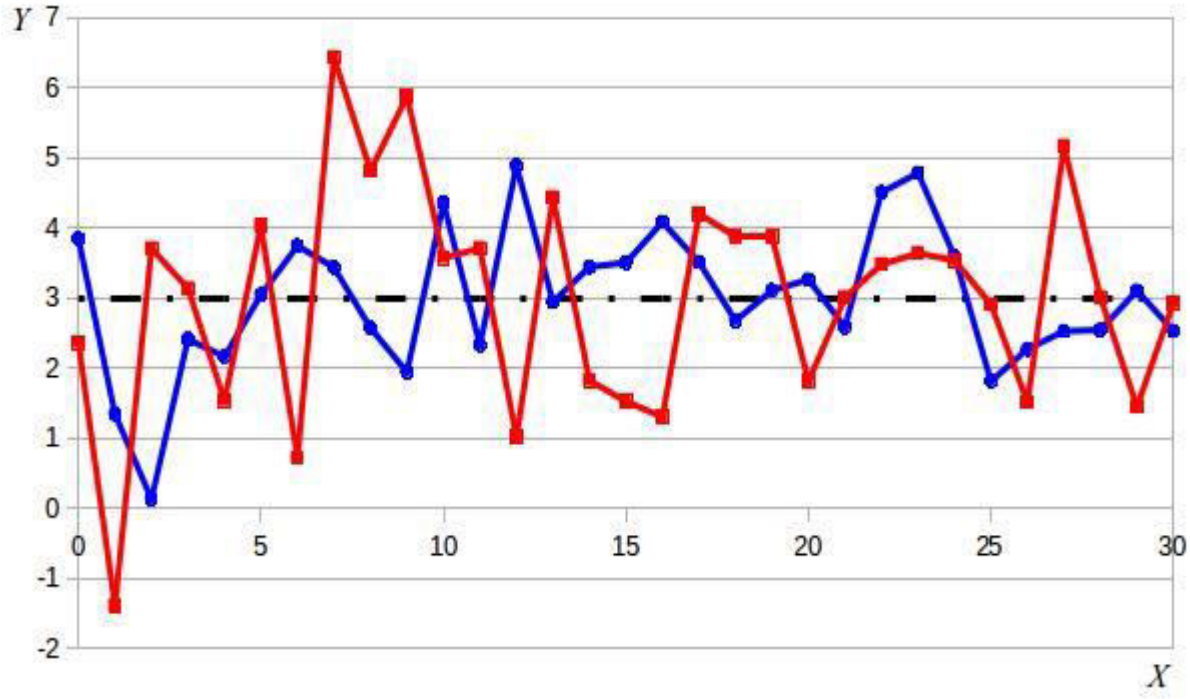


Рис.1.1 Дисперсія випадкової величини: σ_2■ =2,58; σ_2● =1,00

Для випадкових величин дисперсія еквівалентна центрованому моменту другого порядку.

Властивості дисперсії:

- Дисперсія будь-якої випадкової величини не від’ємна;

- Якщо дисперсія випадкової величини кінцева, то кінцеве і математичне очікування (середнє значення) випадкової величини;
- Якщо випадкова величина постійна, то її дисперсія дорівнює нулю (вірно також зворотне).

У випадку, коли по необхідно оцінити дисперсію генеральної сукупності по вибірці, для розрахунку використовують трохи іншу формулу

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}$$

У цьому випадку n - обсяг вибірки, за якою оцінюють дисперсію генеральної сукупності.

Корінь квадратний з дисперсії називають стандартним або середньоквадратичним відхиленням:

$$\sigma = \sqrt{\sigma^2}$$

Дисперсія також використовується для обчислення стандартної похибки вибірки обсягом n:

$$SD_x = \sqrt{\frac{\sigma^2}{n}}$$

В даному випадку воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

Середньоквадратичне відхилення - в математичній статистиці — величина, що характеризує стандартне відхилення вибіркового середнього, розраховане по вибірці розміром із генеральної сукупності [18]. Значення середнього квадратичного відхилення середнього арифметичного залежить від дисперсії генеральної сукупності та обсягу вибірки .

Вибірковий розподіл вибіркового середнього утворюється шляхом повторювання експериментів і фіксування щоразу отриманого середнього. Таким чином отримують розподіл різних середніх, і цей розподіл має своє власне середнє та дисперсію. Математично дисперсія отриманого вибіркового розподілу дорівнює дисперсії сукупності, поділеній на обсяг вибірки. Це тому, що за збільшення обсягу вибірки вибіркове середнє скупчується ближче до середнього сукупності.

Отже, співвідношення між середнім квадратичним відхиленням середнього арифметичного і стандартним відхиленням буде таким, що для даного обсягу вибірки середнє квадратичне відхилення середнього арифметичного дорівнює стандартному відхиленню, поділеному на квадратний корінь від обсягу вибірки. Іншими словами, середнє квадратичне відхилення середнього арифметичного є мірою розсіяння вибіркових середніх довкола центру розподілу сукупності.

У регресійному аналізі, термін "середнє квадратичне відхилення середнього арифметичного" відноситься або до квадратного кореня із скороченого критерію хі-квадрат або середнього квадратичного відхилення середнього арифметичного конкретного коефіцієнту регресії (як це використовується, наприклад, в довірчих інтервалах).

Середнє квадратичне відхилення середнього арифметичного іноді називають "стандартною помилкою" або "стандартною похибкою". Ці терміни є неоднозначними і не рекомендуються до використання як такі, що можуть призвести до плутанини.

В даному випадку воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

Коефіцієнт варіації — відносна величина, що служить для характеристики розсіяння (мінливості) ознаки [19]. Являє собою

відношення середнього квадратичного відхилення S до середнього арифметичного , виражається у відсотках.

Коефіцієнт варіації застосовується тоді, коли необхідно порівняти мінливість ознак об'єкта, які виражені в різних одиницях вимірювання. Має зміст винятково для величин, які вимірюються у шкалах відношень.

Мінливість вважається слабкою, якщо ν<10%; якщо ν від 11-25%, то середньою і значною за ν>25%.

В даному випадку воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

Коефіцієнт осциляції – характеризує відносне коливання крайніх значень ознаки навколо середньої [20]. Вимірюється за допомогою формули:

$$V_r = \frac{R}{x_{\text{сер}}} * 100;$$

В даному випадку воно використовується потім для порівняльного аналізу результатів роботи тестової вибірки часових характеристик з усією сукупністю (генеральною сукупністю даних).

1.5 Огляд програмних аналогів

1.5.1 RapiTime

RapiTime – це інструмент [21] для автоматизованого часового аналізу програм. RapiTime призначений для систем які є критичними до часу виконання програм незалежно від їхнього розміру. Створений для ефективної розробки та відлагодження програм призначених для автомобільної, ракето-космічної, авіаційної промисловості та в робототехніці. RapiTime знижує загальні витрати, необхідні для перевірки часу, оптимізує програмне забезпечення, яке підпадає під оновлення та інтегрує критичні вбудовані системи реального часу.

RapiTime вимірює і отримує відомості про час виконання заданого користувачем фрагменту коду за допомогою часових замірів. Вхідними даними програми є набір вихідних файлів мови програмування C або Ada чи виконуваний файл. Результати часових характеристик роботи фрагментів програми об’єднуються відповідно до розробленої структури програмного забезпечення, для того щоб коректно визначити час роботи всієї програми.

Аналіз часових характеристик програми можна реалізовувати в різних аспектах, що дозволяє аналізувати кожний виклик функції або звернення до певної частини коду окремо. Програма дозволяє також задати рівень деталізації аналізів контекстів регулюється за допомогою анотацій.

RapiTime надає можливість обробляти ітерації циклів за допомогою спеціального розвернення цих циклів. Для кожного такого контексту циклу, межа циклів визначається за допомогою часових замірів (або з анотацій).

RapiTime не тільки обчислює оцінку часових характеристик роботи програми як єдине (ціле) число, а також весь розподіл ймовірностей часу виконання найдовшого шляху в програмі (та інших підпрограмах). Цей розподіл має окремі ділянки (абсолютна верхня межа, нижня межа, узагальнююча середня межа).

Визначення часових характеристик за допомогою RapiTime складається з наступних дій:

1) під час виконання програми на системі, де вона була розроблена будується трейс виконання. Трейс – це послідовність часових замірів програми що показує частини програми які є активними в даний момент;
2) для того щоб створити трейс виконання, вихідний код повинен бути попередньо оброблений (спеціальними точками інструментальних засобів або за допомогою iPoints), які будуть вказувати який фрагмент коду буде виконуватися;

3) RapiTime отримує показники часової ефективності роботи програми для кожного заданого фрагменту коду та/або функцій чи класів;
4) після отримання часових характеристик роботи програми RapiTime розроблює спеціальний звіт де вказує які частини програми можна оптимізувати для покращення часових характеристик та оптимізації коду.

Програмна працює з наступними процесорами:

- ARM (ARM7, ARM9, ARM10, ARM11, Cortex-M, Cortex-R, Cortex-A);
- Intel (x86, Pentium, Atom);
- IBM (PowerPC G1, G2, G3, 401, 403, 405, 440);
- Texas Instruments (MSP430, TMS320, TMS570, C2000);
- Infinion (XE161, XE162, XE164, XE167, XE169, TriCore);
- Renesas (V850, V850E, RH850);
- Motorola processors (MPC555, HCS12).

Перелік компіляторів, які підтримує система:

- ANSI C;
- Arm DS-5;
- Borland;
- CodeWarrior HCS12;
- Cosmic;
- Diab;
- GNU GCC;
- Green Hills;
- IAR;
- Keil C51;

- TASKING;
- Visual Studio.

Головне вікно програми RapiTime зображено на рис. 1.1.

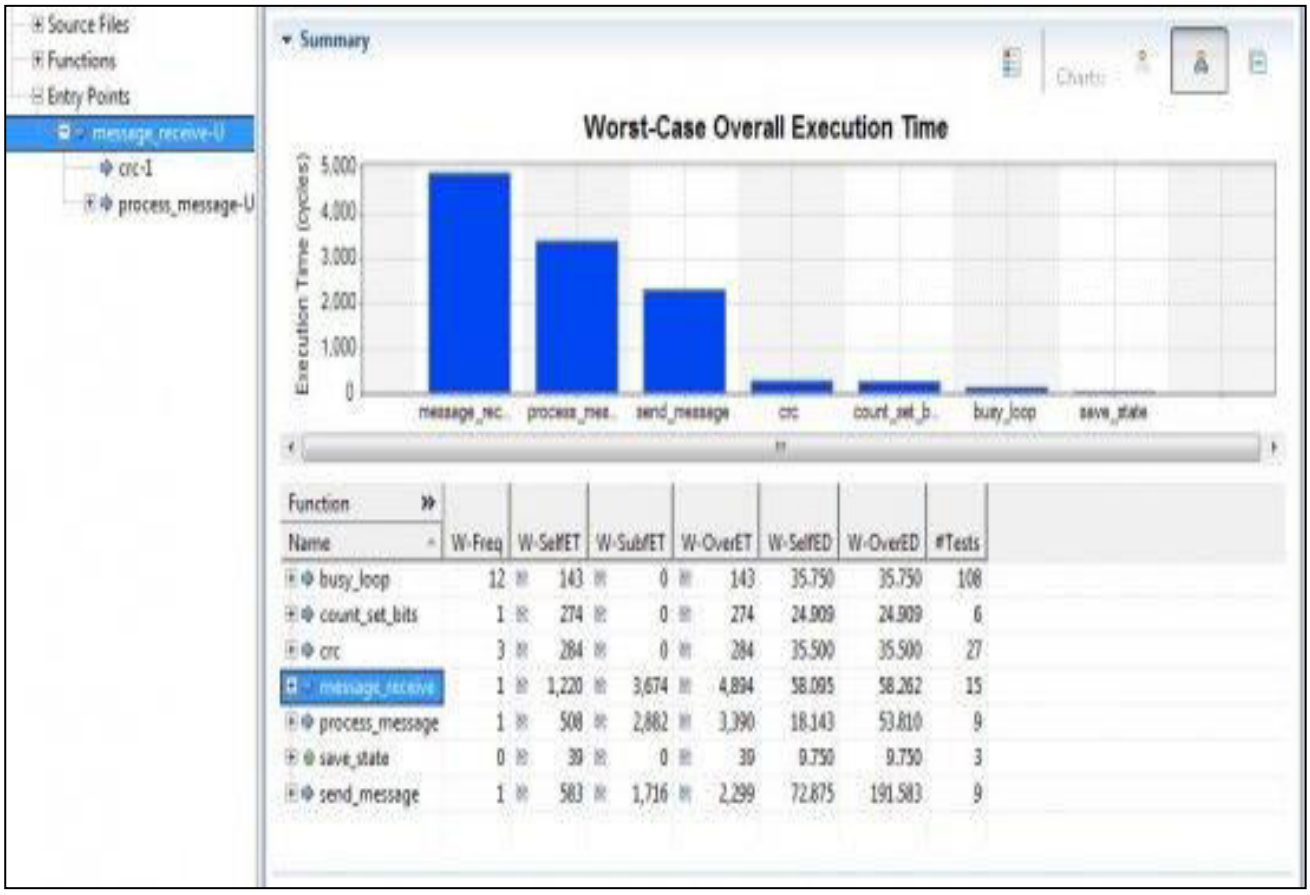
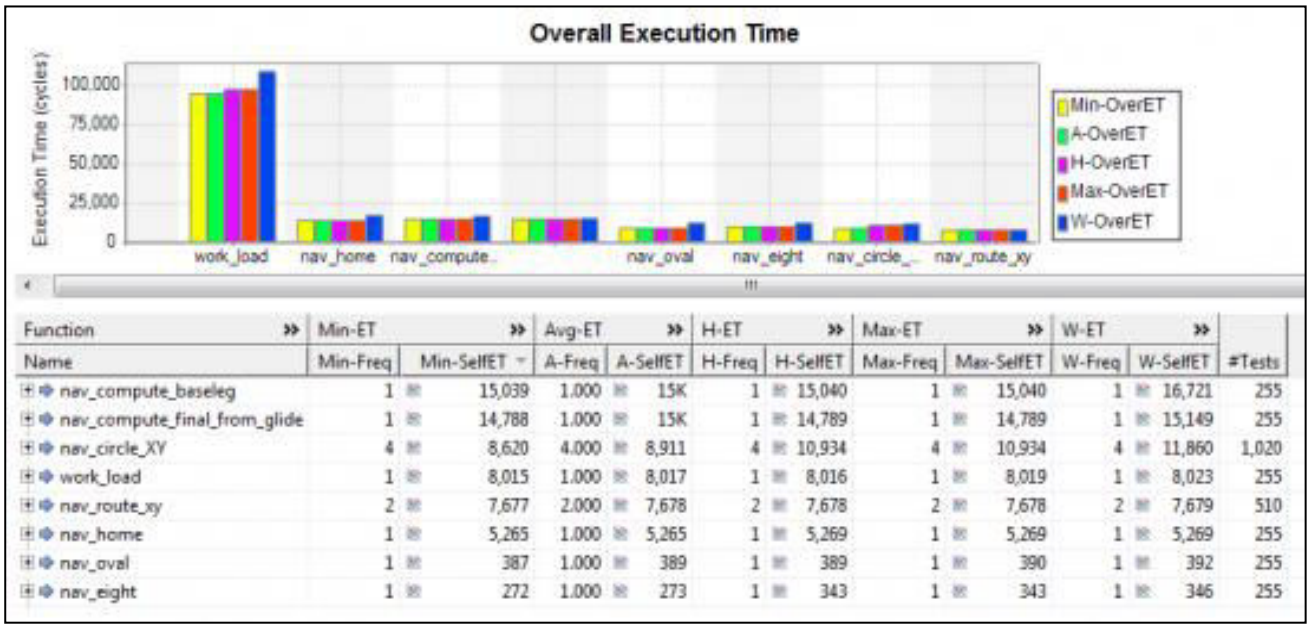


Рисунок 1.1 – Головне вікно RapiTime

Приклад визначення часових показників роботи програми зображено на рис. 1.2.



Function	Min-ET		Avg-ET		H-ET		Max-ET		W-ET		
Name	Min-Freq	Min-SelfET	A-Freq	A-SelfET	H-Freq	H-SelfET	Max-Freq	Max-SelfET	W-Freq	W-SelfET	#Tests
nav_compute_baseleg	1	15,039	1.000	15K	1	15,040	1	15,040	1	16,721	255
nav_compute_final_from_glide	1	14,788	1.000	15K	1	14,789	1	14,789	1	15,149	255
nav_circle_XY	4	8,620	4.000	8,911	4	10,934	4	10,934	4	11,860	1,020
work_load	1	8,015	1.000	8,017	1	8,016	1	8,019	1	8,023	255
nav_route_xy	2	7,677	2.000	7,678	2	7,678	2	7,678	2	7,679	510
nav_home	1	5,265	1.000	5,265	1	5,269	1	5,269	1	5,269	255
nav_oval	1	387	1.000	389	1	389	1	390	1	392	255
nav_eight	1	272	1.000	273	1	343	1	343	1	346	255

Рисунок 1.2 – Вікно часових показників роботи програми в RapiTime

1.5.2 Аналізатор aiT

Програмний продукт aiT від компанії AbsInt являє собою інструмент для визначення оцінки часу виконання програм в операційних системах реального часу [22]. aiT був розроблений для проектування систем реального часу з суворими вимогами до забезпечення безпеки, а також для систем де час виконання програми відіграє важливу роль (виробничі лінії, мікроконтролери вбудованих систем та ін). Дане програмне забезпечення було створено в тісній кооперації з компанією розробником літаків Airbus, з метою забезпечення безпеки під час створення системи автоматичного керування літальними засобами.

Для визначення безпечної верхньої межі часу виконання програми, aiT статично досліджує систему реального часу на основі формальних моделей кеша і конвеєра відповідного мікропроцесора і автоматично визначає граничні часові показники кожної задачі. Він аналізує всі бінарні файли програми і враховує вплив конвеєра та кешу мікропроцесора на час її виконання.

Визначення часової оцінки програми в системі aiT складається з такої послідовності:

1) реконструкція потоку всіх бінарних файлів програмного забезпечення
2) аналіз значень (визначення діапазонів адрес в пам’яті)
3) аналіз кешу (класифікація посилань на пам'ять на можливість звернення до кеша)
4) аналіз конвеєра (прогнозування поведінки програми на процесорному конвеєрі)
5) аналіз шляхів (визначення найгіршого шляху виконання програми)
6) аналіз циклів та рекурсивних процедур

Перелік підтримуваних процесорів і компіляторів наведений в таблиці 1.1.

Таблиця 1.1 – Компілятори та процесори, що підтримує система

Процесор	Компілятори
Am486, IntelDX4	CAD-UL Tool Suite compiler
ARM	ARM CompCert (INRIA/AbsInt) GCC Green Hills MULTI for ARM, C або Ada IAR Keil MDK-ARM TASKING (Altium) TI (Texas Instruments)
C16x/ST10	TASKING (Altium)

	KEIL (ARM)
C28x	TI (Texas Instruments)
C33	TI (Texas Instruments)
ERC32	GCC GNAT
HCS12	Hiware (Metrowerks/Freescale) Cosmic IAR
i386DX	PL/I compiler
LEON2, LEON3	GCC GNAT
M68020	HP 68000, C або Ada XD Ada (EDS) GCC
PowerPC 5xx, e200 (55xx, 56xx, 57xx), e300 (603e, 82xx, 83xx, 52xx), 7448, 7448s, 750, 755, 755s	Diab (WindRiver), C або Ada GHS (Green Hills), C або Ada GCC HighTec GCC CodeWarrior (Freescale)
TriCore	TASKING (Altium) GCC HighTec GCC

	Diab (WindRiver)
V850E	GHS (Green Hills)

Основні переваги продукту:

- результати аналізів розроблюються для всієї тестованої системи та є дійсними для всіх варіантів роботи програмного забезпечення що може гарантувати правильність роботи системи;
- дозволяє значно зменшити час на розробку за рахунок автоматизації тестів та вимірювань роботи програми;
- зручний інтерфейс, який візуально зрозумілий користувачу і дозволяє досліджувати стани конвеєра і кеша в будь-якій точці виконання тієї чи іншої задачі.
- система аналізує не вхідний код написаний на мові високого рівня, а бінарні файли системи, що дозволяє не вносити зміни в програму на етапі розробки та компіляції програмного забезпечення.

Головне вікно aiT зображено на рис 1.3.

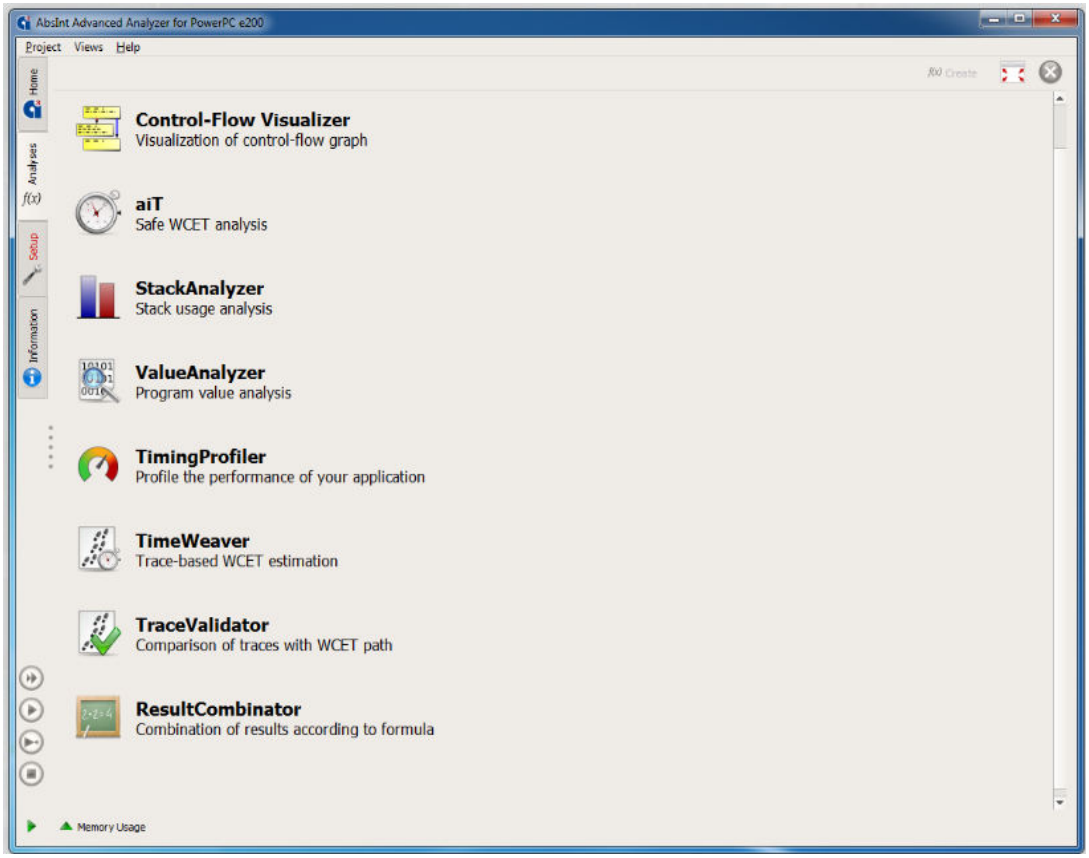


Рисунок 1.3 – Головне вікно aiT

Приклад візуалізації структури програмного забезпечення в aiT зображено на рис. 1.4.

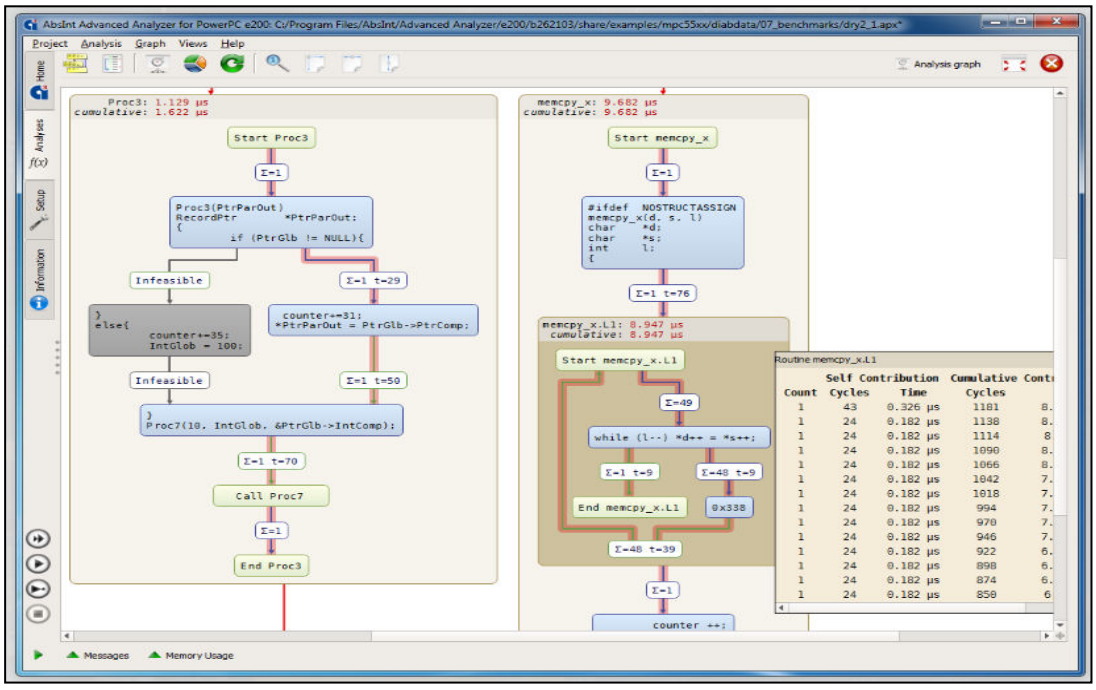


Рисунок 1.4 – Візуалізація структури програмного забезпечення в aiT

Приклад часової оцінки функцій програми в aiT зображено на рис. 1.5.

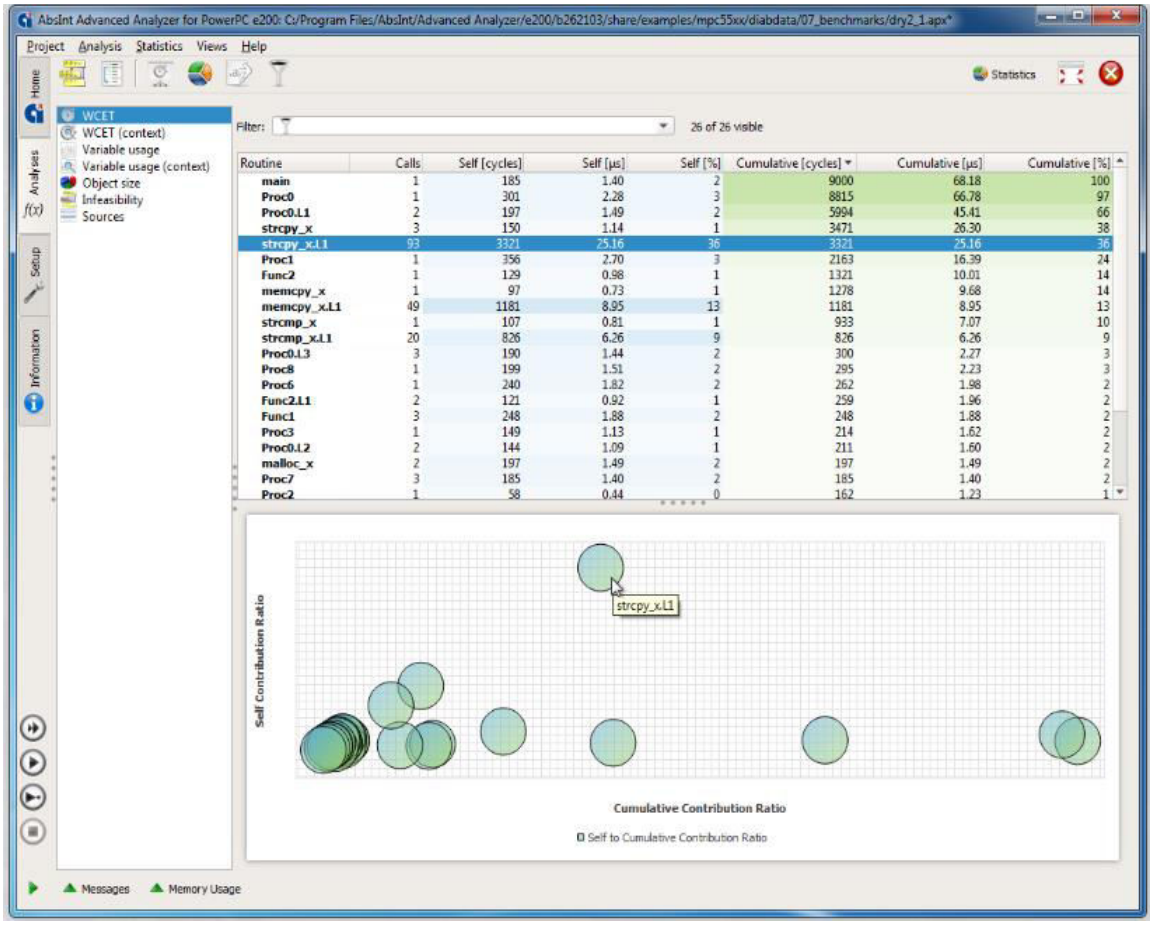


Рисунок 1.5 – Часова оцінка функцій програми

2 ОБГРУНТУВАННЯ МЕТОДУ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ЗАСОБІВ СИНХРОНІЗАЦІЇ ПОТОКІВ

Завдання розробки нових ефективних методів та засобів синхронізації потоків не втратило своєї актуальності. Це майже неможливо зробити без аналізу результатів роботи вже існуючих примітивів синхронізації та можливостей їх застосування. Особливо це стосується операційних систем реального часу, які на сьогоднішній день займають домінуюче положення в сфері розробки автономних програмних систем.

Задачі, які вирішують більшість автономних систем є такими:

- задачі жорсткого реального часу (коли перевищення часу виконання поставлених завдань може призвести до невідворотних наслідків);
- задачі м’якого реального часу (коли перевищення часу вирішення небажане, але припустиме);
- задачі некритичні до часу виконання.

Для перших двох типів задач втрата «зайвого часу», має фатальні наслідки. Через неоптимальний вибір примітиву синхронізації або можливості його застосування зростає загальний час виконання програм, резервуються зайві ресурси системи, зростає найгірший час виконання програм. Відповідно дослідження щодо ефективного з точки зору часових характеристик примітиву синхронізації в залежності від зростання складності задачі дозволить в майбутньому розробляти програмне забезпечення з більш оптимальною реалізацією.

2.1 Аналіз операційних систем

2.1.1 VxWorks

VxWorks - вбудована операційна система жорсткого реального часу, що застосовується в пристроях з підвищеними вимогами до продуктивності і безпеки [71]. Її міць, надійність і компактність дозволяють розробникам

оперативно створювати функціональні додатки з високим рівнем якості і при оптимальних витратах.

Для спрощення рішення типових задач також існують галузеві пакети технологій на базі VxWorks, орієнтовані на конкретні вертикальні ринки – наприклад, платформи для промислової автоматизації, мереж і телекомунікацій, електроніки, а також сертифіковані платформи для відповідальних систем.

Особливості та переваги VxWorks:

- Домени захисту пам'яті, що можна налаштовувати;
- "Жорсткий" реальний час: перемикання контексту / реакція на переривання - одиниці мкс;
- Підтримка POSIX API;
- Достатня для розробки кількість ОЗУ / ПЗУ - сотні КБ;
- Підтримка багатоядерних процесорів;
- Розширена підтримка мереж TCP / IP (IPv4, IPv6)
- Інтеграція з SCADA-додатками на базі Windows і промисловими мережами;
- Підтримка веб-сервісів (XML, SOAP, WSDL) ;
- Функції управління енергоспоживанням;
- Потужний графічний пакет Tilcon Graphics Suite;
- Підтримка баз даних: реляційні СУБД і БД реального часу;
- Потужна інтегроване середовище розробки на базі Eclipse;
- Інтеграція із засобами внутрішньо схемного налагодження (JTAG) Wind River
- Робота з процесорами: x86, ARM, MIPS, PowerPC, ColdFire
- Широка підтримка обладнання Advantech, MEN Mikro Elektronik, ADLINK, LiPPERT, RTD, ADDI-DATA, Hilscher, Diamond Systems і інших виробників

- Сертифікація IEC 15408 ("Загальні критерії") EAL 4/4 + / 6 +, DO-178B рівень A, МЕК 61508 SIL 3, CENELEC EN 50128 і FDA 510 (k)
- Відкритий вихідний текст, можливість побудови ОС з вихідних текстів

Типові застосування: відповідальні системи, авіація / космонавтика, промислові додатки, мережі / телекомунікації, медичне приладобудування.

Головний акцент розробники системи роблять на надійність і відмовостійкість. Саме тому дана операційна система стала стандартом для всієї космічної електроніки NASA. VxWorks виробництва компанії Wind River застосовується в місіях NASA Pathfinder, Deep Space One, Mars Odyssey, Stardust в супутнику PROBA Європейського Космічного Агенства ESA і на човнику Lifeboat Міжнародної Космічної Станції.

До недоліків системи можна віднести її повільний розвиток. Основний акцент робиться на старі перевірені програми і драйвери. Виникають проблеми з підтримкою нового обладнання.

Це досить дорога операційна система. Вибір її виправданий або при великих обсягах продукції, що випускається або в системах, що вимагають високої надійності

2.1.2 QNX

Починаючи з 1980 р, розробники використовували і покладалися на технологію QNX при побудові систем, що вимагають безвідмовного функціонування: медичних приладів, телематичних пристроїв, Інтернет-маршрутизаторів, call-центрів, служби 911, систем управління технологічними процесами, навіть систем управління для Міжнародної Космічної Станції. Незалежно від їх масштабу і складності, ці системи об'єднує одне: всі вони працюють безперервно по 24 години в день, 365 днів у році, без перерв.

QNX - операційна система жорсткого реального часу, розроблена спеціально для високовідповідальних додатків, які функціонують роками без збоїв у роботі. Надійність - QNX забезпечується її архітектурою - це справжня операційна система на основі мікроядра. В операційній системі QNX ядром обробляються тільки базові примітиви ОС (сигнали, таймери, планування). Всі інші компоненти - драйвери, файлові системи, стеки протоколів прикладні програми - виконуються поза межами ядра як окремі процеси, кожен в своєму захищеному адресному просторі. Такий підхід автоматично забезпечує системам на основі QNX "вбудовану" відмовостійкість. Всі компоненти QNX використовують для спілкування один з одним єдиний, чітко детермінований механізм - обмін повідомленнями. Він утворює між компонентами системи віртуальну "програмну шину", що дозволяє підключати до неї або, навпаки, відключати будь-який компонент "миттєво". Повідомлення можуть вільно передаватися між вузлами обчислювальної мережі, надаючи прозорий доступ до будь-якого ресурсу, де б він не знаходився.

Використання операційної системи QNX дозволяє:

- Створювати системи, здатні до самовідновлення - в QNX будь-який компонент в разі відмови може бути перезапущений динамічно, не порушуючи роботу мікроядра і інших компонентів.
- Використовувати одну й ту ж саму ОС у всій лінійці продуктів.
- Виробляти оновлення системного програмного забезпечення без зупинки роботи кінцевого пристрою.
- Використовувати підтримку журналу зміни файлів (ким і які зміни внесені).

2.2 Аналіз середовищ розробки для досліду

2.2.1 Eclipse

Eclipse вільне інтегроване середовище розробки модульних кроссплатформених додатків. Розвивається і підтримується Eclipse Foundation.

Найбільш відомі програми на основі Eclipse Platform - різні «Eclipse IDE» для розробки ПЗ на багатьох мовах.

Eclipse служить в першу чергу платформою для розробки розширень, чим він і завоював популярність: будь-який розробник може розширити Eclipse своїми модулями. Вже існують Java Development Tools (JDT), C / C ++ Development Tools (CDT), що розробляються інженерами QNX спільно з IBM, і додатки для мов Ada (GNATbench, Hibachi), COBOL, FORTRAN, PHP, X10 (X10DT) та ін. Безліч розширень доповнює середовище Eclipse диспечерами для роботи з базами даних, серверами додатків і ін.

Eclipse JDT (Java Development Tools) - найбільш відомий модуль, націлений на групову розробку: середовище інтегроване з системами управління версіями - CVS, GIT , для інших систем (наприклад, Subversion, MS SourceSafe) існують плагіни. Також пропонує підтримку зв'язку між IDE і системою управління завданнями (помилками). В основній версії постачання включена підтримка трекера помилок Bugzilla, також є безліч розширень для підтримки інших трекерів (Trac, Jira та ін.). В силу безкоштовності і високої якості, Eclipse в багатьох організаціях є корпоративним стандартом для розробки додатків.

Eclipse написана на Java, тому є платформо-незалежним продуктом, за винятком бібліотеки SWT, яка розробляється для всіх поширених платформ. Бібліотека SWT використовується замість стандартної для Java бібліотеки Swing. Вона повністю спирається на базову лінію у платформу (операційну систему), що забезпечує швидкість і натуральний зовнішній вигляд призначеного для користувача інтерфейсу, але іноді викликає на різних платформах проблеми сумісності і стійкості додатків.

2.2.2 GNU Compiler Collection

GNU Compiler Collection – набір компіляторів для різних мов програмування, розроблений в рамках проекту GNU [72]. GCC є вільним

програмним забезпеченням, поширюється фондом вільного програмного забезпечення (FSF) на умовах GNU GPL і GNU LGPL і є ключовим компонентом GNU toolchain. Він використовується як стандартний компілятор для вільних UNIX-подібних операційних систем.

Спочатку названий GNU C Compiler підтримував тільки мову Ci. Пізніше GCC був розширений для компіляції вихідних кодів на таких мовах програмування, як C ++, Objective-C, Java, Фортран, Ada, Go, GAS і D.

Зовнішній інтерфейс GCC є стандартом для компіляторів на платформі UNIX. Користувач викликає керуючу програму, яка називається gcc. Вона інтерпретує аргументи командного рядка, визначає і запускає для кожного вхідного файлу свої компілятори потрібної мови, запускає, якщо необхідно, асемблер і компонувальник.

Компілятор кожної мови є окремою програмою, яка отримує вихідний текст і породжує висновок на мові асемблера. Всі компілятори мають загальну внутрішню структуру: front end, який виробляє синтаксичний розбір і породжує абстрактне синтаксичне дерево, і back end, який конвертує дерево в Register Transfer Language (RTL), виконує різні оптимізації, потім породжує програму на мові асемблера, використовуючи архітектурно-залежне зіставлення зі зразком.

GCC є громіздким, повільним і генеруючим низькоякісний код. Унаслідок такого критичного ставлення, а також через то що він досить обмежує (в порівнянні з BSD) ліцензії GPL, під якою випущена колекція компіляторів, була зроблена спроба замінити GCC іншими компіляторами, наприклад, PCC.

2.3 Аналіз примітивів синхронізації для досліду

2.3.1 Умовна змінна

Умовна змінна - примітив синхронізації, що забезпечує блокування одного або декількох потоків до моменту надходження сигналу від іншого потоку про виконання деякого умови або до закінчення максимального проміжку часу

очікування [73]. Умовні змінні використовуються разом з асоційованими м'ютексами і є елементом деяких видів моніторів.

2.3.2 Sleepons

Sleepon дуже схожі на умовні змінні, з невеликими відмінностями. Як і умовні змінні,Slepopon (pthread_sleepon_lock ()) можна використовувати для блокування, поки умова не стане істиною (наприклад, значення місця, що змінюється в пам'яті). Але на відміну від умовних зміних, які повинні бути виділені для кожної умови, яку потрібно перевірити, Slepoon блокує свою функціональність у мультиплексному режимі через єдиний мьютекс та динамічно виділену умовну змінну, незалежно від кількості умов, що перевіряються. Максимальна кількість умовних змінних у підсумку дорівнює максимальній кількості заблокованих потоків. Ці примітиви мають шаблон, який зазвичай використовуються в ядрі UNIX [74].

2.3.3 Семафор

Семафор – примітив синхронізації роботи процесів і потоків, в основі якого лежить лічильник, над яким можна проводити дві атомарні операції: збільшення і зменшення значення на одиницю, при цьому операція зменшення для нульового значення лічильника є блокуючою [75]. Служить для побудови більш складних механізмів синхронізації і використовується для синхронізації паралельно працючих задач, для захисту передачі даних через пам'ять, що розділяється, для захисту критичних секцій, а також для управління доступом до апаратного забезпечення. Обчислювальні семафори використовуються для контролю над обмеженими ресурсами. Двійкові семафори забезпечують взаємне виключення виконання критичних секцій, а їх спрощеною реалізацією є м'ютекси, які більш обмежені у використанні. Крім взаємного виключення в загальному випадку семафори і м'ютекси можуть використовуватися в безлічі інших типових алгоритмів, включаючи сигналізування іншим задачам, дозвіл проходження певних контрольних точок тільки для однієї задачі одноразово по аналогії з

турнікетом, завдання виробника і споживача, що має на увазі передачу даних від одних завдань іншим, бар'єри, що дозволяють синхронізувати групи завдань в певних контрольних точках, умовні змінні для оповіщення інших завдань про які-небудь події і блокування читання і записи, що дозволяють одночасне читання даних, але забороняють їх одночасну зміну.

Типовими проблемами використання семафорів є одночасне блокування двох завдань в очікуванні один одного і ресурсне голодування, в результаті чого ресурс може бути періодично недоступний для одних завдань через його використання іншими завданнями. При використанні в процесах з пріоритетом реального часу може виникнути інверсія пріоритетів, яка може привести до необмеженого за часом блокування процесу з більш високим пріоритетом через захоплення семафора процесом з більш низьким пріоритетом, в той час як процесорний час віддається процесу із середнім пріоритетом, рішенням чого є успадкування пріоритетів.

2.3.4 М’ютекс

М'ютекс – примітив синхронізації, що забезпечує взаємне виключення виконання критичних ділянок коду [76]. Класичний м'ютекс відрізняється від довічного семафора наявністю ексклюзивного власника, який і повинен його звільняти.

Умовно класичний м'ютекс можна представити у вигляді змінної, яка може перебувати в двох станах: у заблокованому і в незаблокованому. При вході в свою критичну секцію потік викликає функцію перекладу м’ютекса в їхній заблокований статус, при цьому потік блокується до звільнення м’ютекса, якщо інший потік вже володіє ним. При виході з критичної секції потік викликає функцію перекладу м’ютекса в незаблокований стан. У разі наявності кількох заблокованих по м'ютексу потоків під час розблокування м’ютекса вибирається довільний з них.

Завданням м’ютекса є захист об'єкта від доступу до нього інших потоків, відмінних від того, який заволодів м'ютексом. У кожений конкретний момент

тільки один потік може володіти об'єктом, захищеним м'ютексом. Якщо іншому потоку буде потрібен доступ до даних, захищених м'ютексом, то цей потік блокується до тих пір, поки м'ютекс не буде звільнений. М'ютекс захищає дані від пошкодження в результаті асинхронних змін.

2.4 Аналіз засобів часових замірів роботи примітивів синхронізації

Часові заміри роботи примітивів синхронізації було здійснено за допомогою системної функції ClockCycles() результат роботи якої являє собою число тактів (періодів частоти) з моменту завантаження до точки, де було здійснено виклик функції. Отриманий результат є величиною наносекундного діапазону та напряму залежить від тактової частоти обраного процесора.

2.5 Аналіз елементів статистичної обробки часових характеристик

Статистична обробка часових характеристик роботи примітивів синхронізації та програми включає в себе пошук та обчислення відповідних математичних величин:

- Медіана;
- Мінімум;
- Максимум;
- Розмах варіації;
- Середне лінійне відхилення;
- Дисперсія;
- Середньоквадратичне відхилення;
- Коефіцієнт варіації;
- Коефіцієнт осциляції;
- Час роботи примітиву синхронізації до загального часу виконання програми порахований у відсотках.

Повний опис даних елементів наводиться у розділі 1.4 цієї роботи.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНСТРУМЕНТАЛЬНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Зовнішнє проектування

3.1.1 Опис функціональних характеристик

Програмна система «QNX C++ SyncPrimComp» реалізує отримання часових характеристик виконання примітивів синхронізації операційної системи QNX та всього часу виконання програми і призначена для отримання часової оцінки роботи заданих примітивів синхронізації на мові C/C++ з використанням програми Excel 2016 для подальшої статистичної обробки та визначення ефективності роботи кожної з порівнюваних груп.

Програмний продукт повинен мати наступні можливості:

- введення і редагування вхідного коду;
- можливість компілювати і запускати вхідний код, за допомогою терміналу операційної системи;
- мінімальне налаштування інтерфейсу терміналу операційної системи;
- вивід на екран розрахованого часу виконання програми та примітиву синхронізації;
- збереження інформації в текстовий файл;
- перенесення відповідних даних в програму Excel 2016 для подальшої статистичної обробки та візуалізації;
- збереження та можливість редагування файлу з розширенням .xlsx програми Excel 2016.

3.1.2 Вхідні дані

Вхідними даними програми, що розробляється, є:

- код на мові C/C++, введений вручну в редакторі;

- текстовий файл, який можна перенести в середовище Excel для подальшої обробки;
- коментарі в коді з додатковою інформацію про вхідні дані програми, що аналізується, час виконання окремих функцій.

3.1.3 Вихідні дані

Результатом роботи програми є наступні вихідні дані:

- вихідний інструментований код C/C++;
- розрахований час виконання примітивів синхронізації, що реалізовані в програмі;
- файл з розширенням .xlsx з результатом статистичної обробки та візуалізації часових характеристик примітивів синхронізації.

3.1.4 Проектування за допомогою UML-діаграм

На початку розробки програми була створена діаграма прецедентів, що є основним етапом при проектуванні системи, бо допомагає визначитися з функціоналом програми, що доступна користувачу [23].

Діаграма прецедентів візуально зображає різноманітні сценарії між акторами (користувачами) і прецедентами (випадками використання), описує функціональні аспекти системи.

Елементи діаграми прецедентів:

- актор – користувач (може бути людина, комп’ютер, організація чи зовнішня система);
- прецедент – випадок використання, дія. Він описує функцію, яку виконуватиме система для досягнення мети користувача. Прецедент має повертати видимий результат, який має значення для користувача системи.

В мові UML [24] є кілька стандартних видів відносин між акторами й варіантами використання:

– відношення асоціації – специфікує семантичні особливості взаємодії акторів і варіантів використання в графічній моделі системи;

– відношення розширення – відзначає той факт, що один з варіантів використання може приєднувати до своєї поведінки деяку додаткову поведінку, певну для іншого варіанту;

– відношення узагальнення – служить для вказівки того факту, що деякий варіант використання А може бути узагальнений до варіанту використання В;

– відношення включення – вказує, що деяка задана поведінка для одного варіанта використання включається як складовий компонента в послідовність поведінки іншого варіанту використання.

В результаті моделювання на етапі проектування було створено діаграму прецедентів (рис. 1.6). Вона включає в себе одного актора-програміста і надає йому наступні можливості:

– введення коду С;

– редагування коду С;

– збереження коду С;

– компіляція і запуск програми на С з використанням терміналу операційної системи;

– збереження результатів в текстовий файл;

– перенесення часових даних в відповідні поля програми Excel 2016 та отримання даних статистичної обробки та візуалізації даних;

– збереження та редагування файлу з розширенням .xlsx.

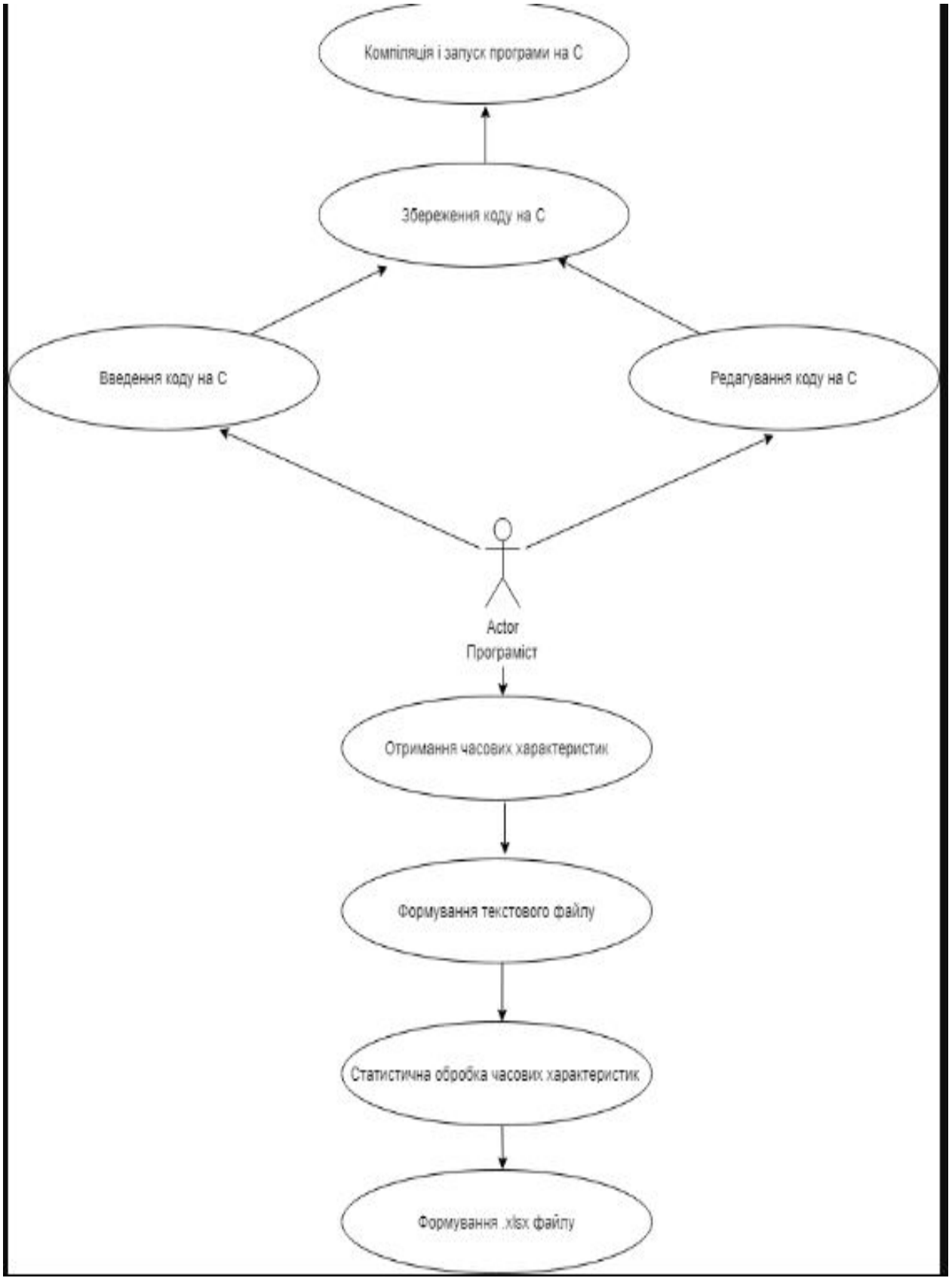

Компіляція і запуск програми на С
Збереження коду на С
Введення коду на С
Редагування коду на С
Actor
Програміст
Отримання часових характеристик
Формування текстового файлу
Статистична обробка часових характеристик
Формування .xlsx файлу

Рисунок 1.6 – Діаграма прецедентів

Діаграма активності – UML-діаграма, на якій показано розкладання деякої діяльності на її складові частини [25]. Під активністю розуміється специфікація виконуваної поведінки у вигляді координовано послідовного і паралельного виконання підлеглих елементів – вкладених об’єктів діяльності та окремих дій, з’єднаних між собою потоками, які йдуть від виходів одного вузла до входів іншого. Діаграми діяльності використовуються для моделювання різних типів процесів.

Діаграма активності складається з обмеженої кількості фігур, з’єднаних стрілками. Основні фігури:

– прямокутники з заокругленнями – дії;

– ромб – рішення;

– широка полоса – початок (розгалуження) і кінець (сходження) витків дії;

– чорний круг – початок процесу (початковий стан);

– чорний круг з обведенням – закінчення процесу (кінцевий стан).

Користувачеві необхідно запустити програмний модуль для отримання часових характеристик роботи програми та обраного примітиву синхронізації в текстовому вигляді та візуальному в термінал операційної системи.

Діаграма активності для отримання часових характеристик зображена на рис. 1.7.

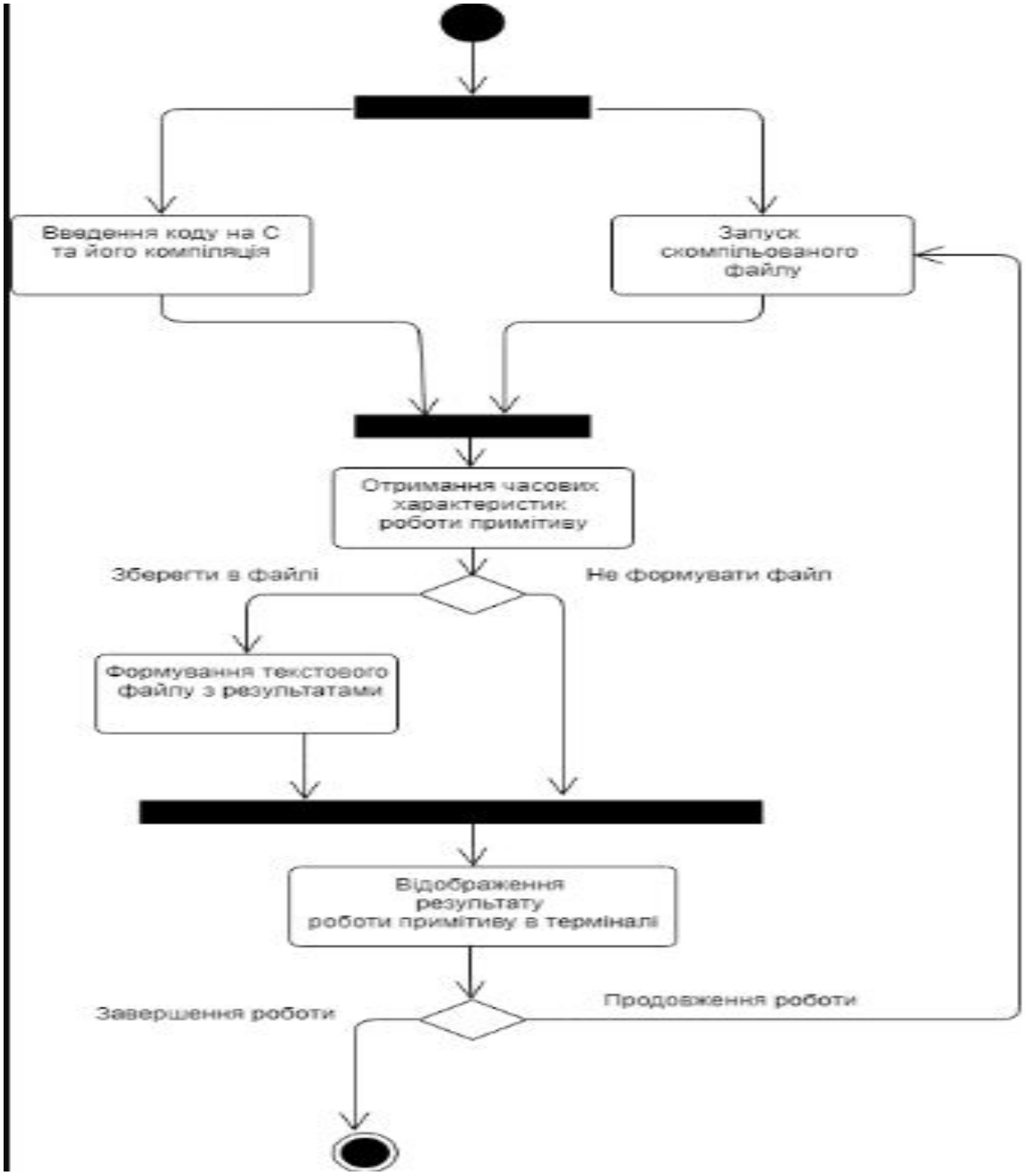


Рисунок 1.7 – Діаграма активності отримання часових характеристик програми

Після того, як користувач отримав текстовий файл з часовими характеристиками роботи програми та примітивів синхронізації, йому необхідно перенести ці дані у відповідні поля програми Excel 2016.

Діаграма активності статистичної обробки часових оцінок роботи програми зображена на рис. 1.8.

Рисунок 1.8 – Діаграма активності статистичної обробки даних

Діаграма станів – це діаграма, що описує можливі послідовності станів і переходів, які в сукупності відображають динамічні аспекти системи [26].

На діаграмі станів використовуються такі умовні позначення:

– коло, що позначає початковий стан;

– коло з маленьким колом усередині, що позначає кінцевий стан;

– скруглений прямокутник, що позначає стан. Верхівка прямокутника містить назву стану. Всередині може бути горизонтальна лінія, під якою записуються активності, що відбуваються в даному стані;

– стрілка, що позначає перехід. Назва події (якщо є), що викликала перехід, відзначається поруч зі стрілкою;

– широка горизонтальна лінія з безліччю вхідних ліній і однією вихідною. Означає об’єднання;

– широка горизонтальна лінія з однією вхідною лінією і безліччю вихідних. Означає розгалуження.

Діаграма станів інтерфейсу [27] програми в процесі отримання часових характеристик роботи програми зображена на рис. 1.9.

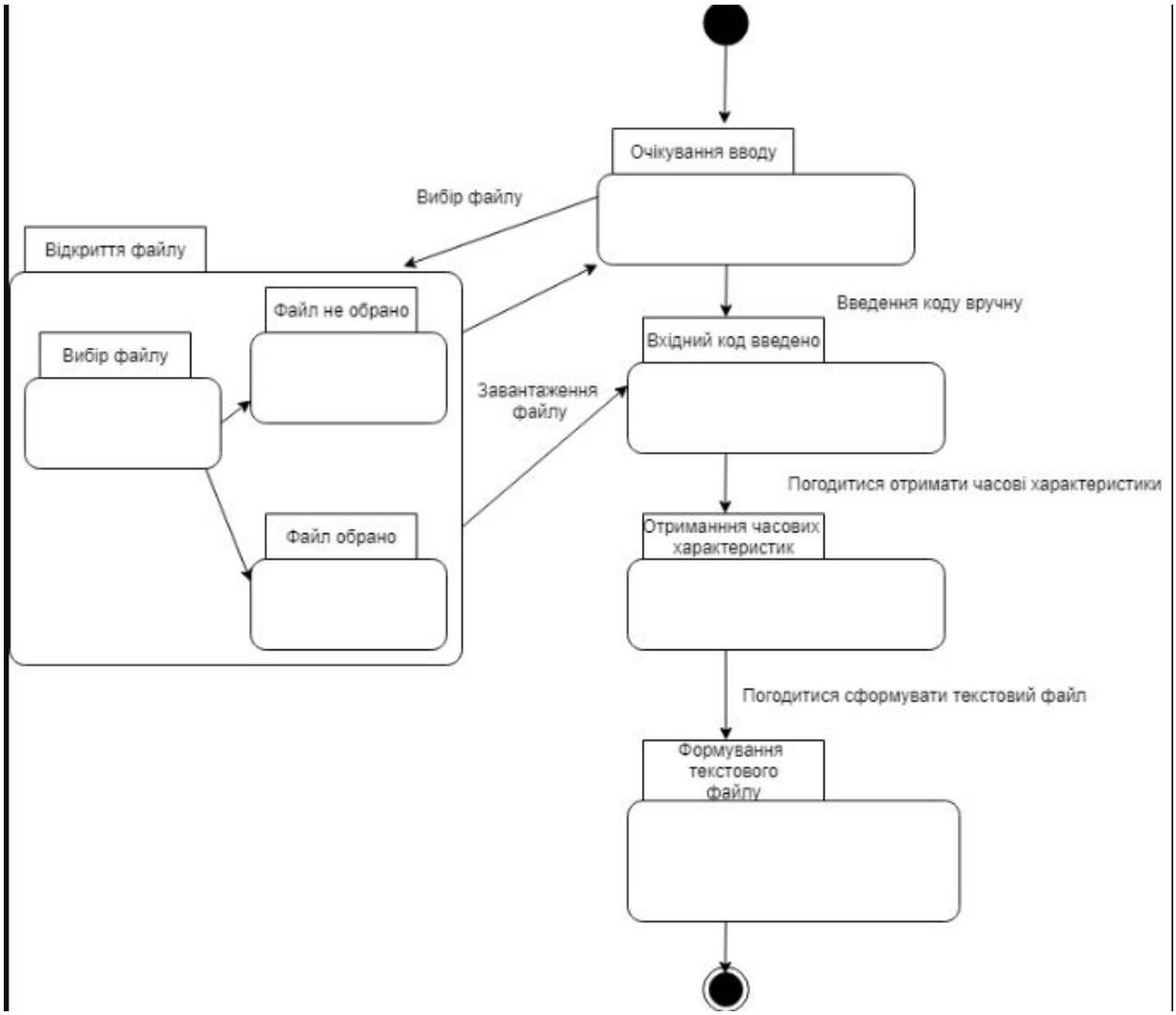


Рисунок 1.9 – Діаграма станів інтерфейсу програми

Діаграма послідовності – діаграма, що відображає взаємодії об'єктів впорядкованих за часом [28]. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень. На діаграмі послідовностей показано у вигляді вертикальних ліній різні процеси або об'єкти, що існують водночас. Надіслані повідомлення зображуються у вигляді горизонтальних ліній, в порядку відправлення.

На рисунку 1.10 зображено діаграму послідовності для типового використання програми користувачем.

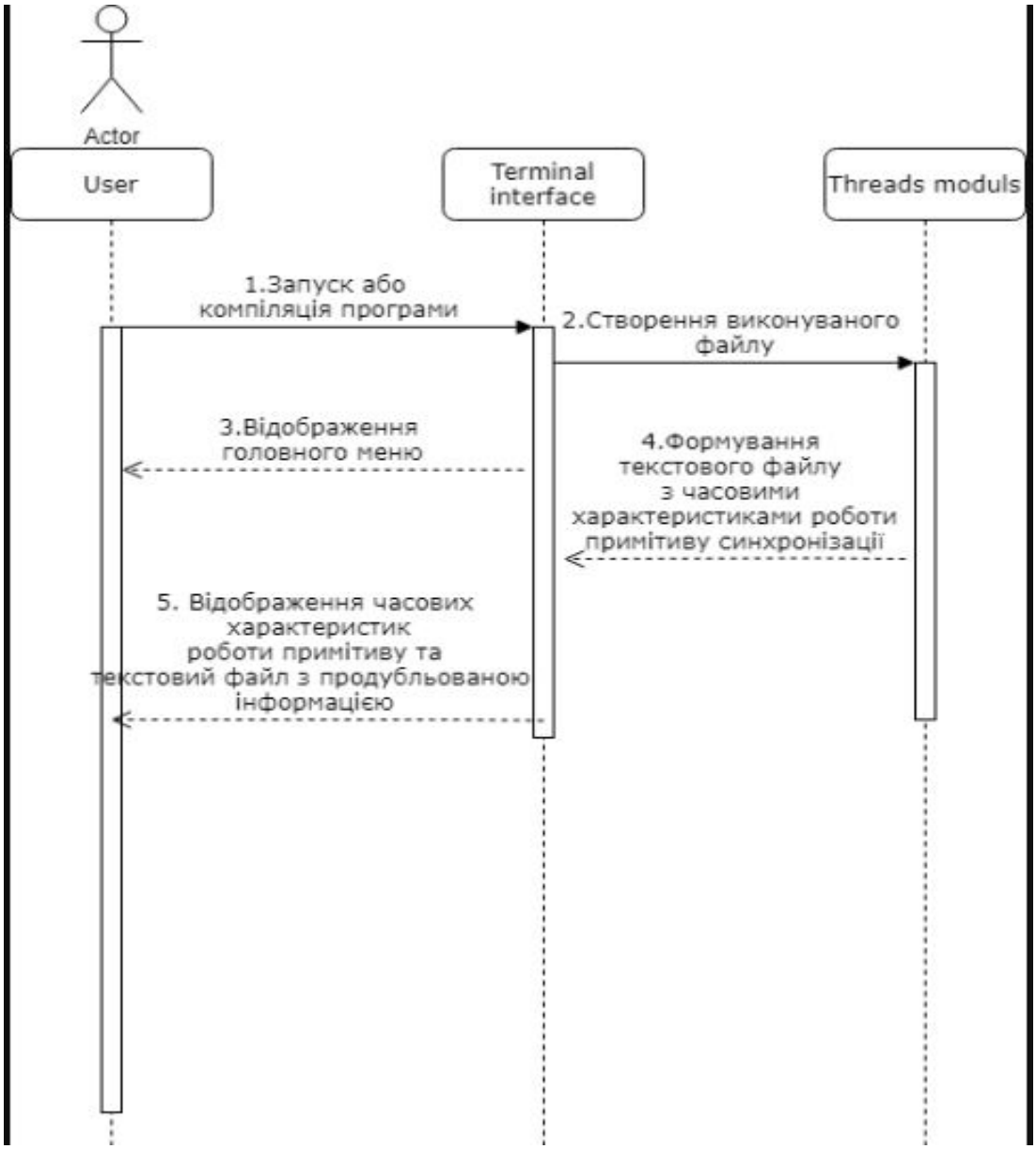


Рисунок 1.10 – Діаграма послідовності типового використання програми

На рисунку 1.11 зображено діаграму послідовності для статистичної обробки даних.

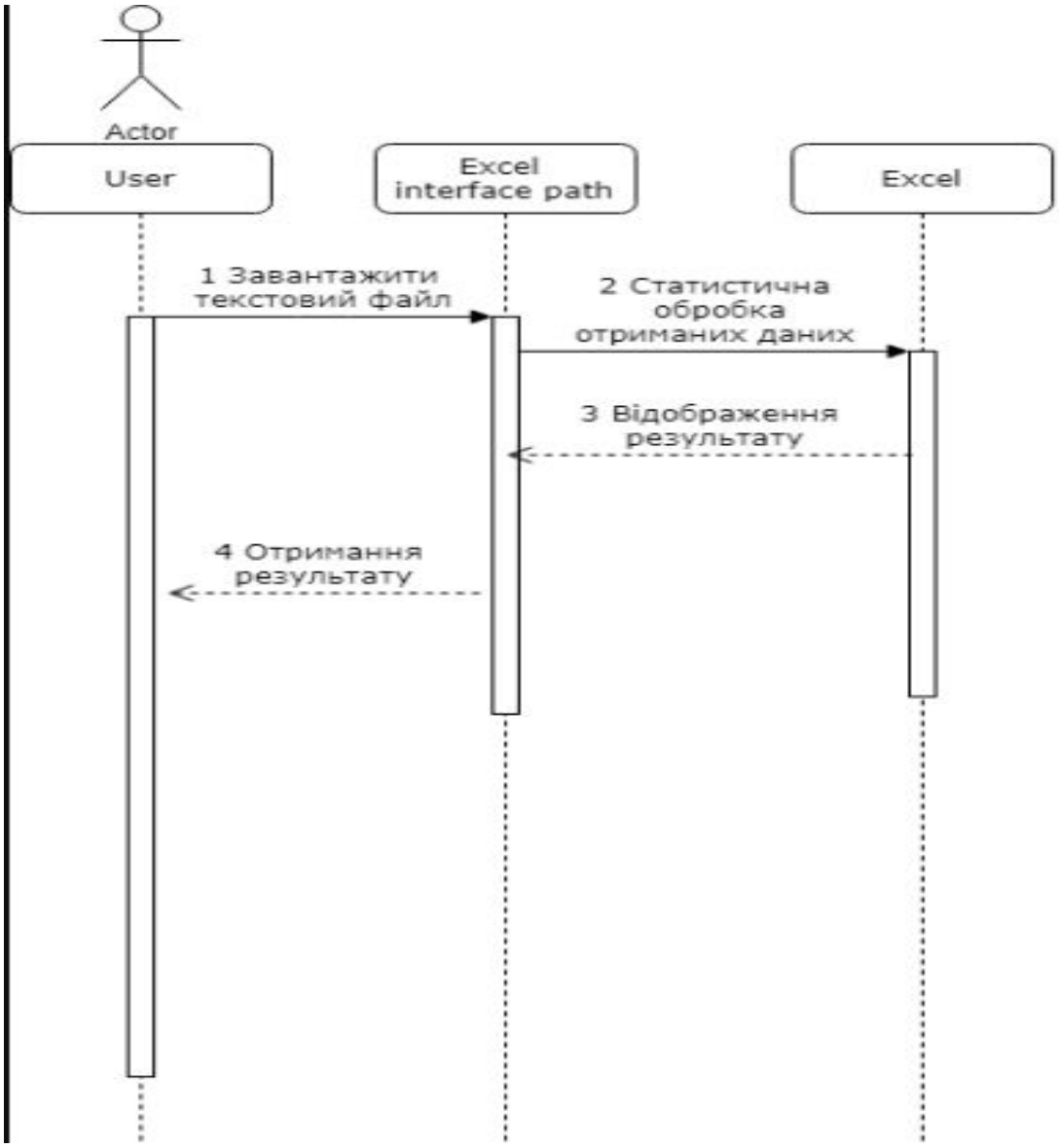


Рисунок 1.11 – Діаграма послідовності статистичної обробки даних

Діаграма компонентів, на відміну від раніше розглянутих діаграм, описує особливості фізичного представлення системи [29]. Діаграма компонентів дозволяє визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами, в ролі яких може виступати початковий, бінарний і виконуваний код. Основними графічними елементами діаграми компонентів є компоненти, інтерфейси і залежності між ними.

Діаграма компонентів розробляється для наступних цілей:

– візуалізація загальної структури початкового коду програмної системи;

– специфікації здійснимого варіанту програмної системи;

– забезпечення багатократного використання окремих фрагментів програмного коду;

– представлення концептуальної і фізичної схем баз даних.

Діаграма компонентів для програмного засобу «QNX C++ SyncPrimComp» представлена на рис. 1.12.

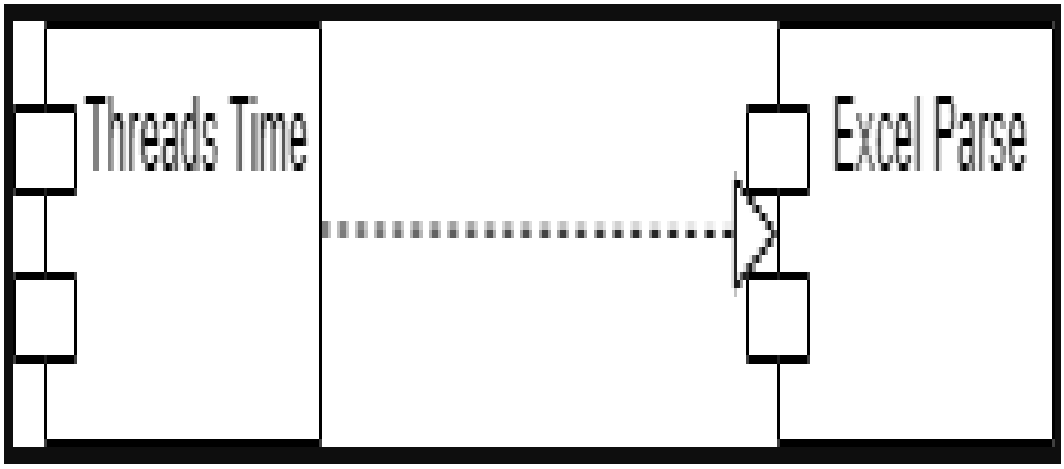


Рисунок 1.12 – Діаграма компонентів

Стрілками на рис. 1.12 показана залежність одних модулів від інших.

Призначення модулів програми:

– Threads Time – формує текстовий файл з часовими характеристиками роботи примітивів синхронізації;

– Excel Parse – модуль, що здійснює статистичну обробку часових

характеристик роботи примітивів синхронізації.

Таким чином, за допомогою UML-діаграм було змодельовано основну функціональність програми, основні активності і процеси. Також, на діаграмі компонентів змодельовану архітектуру майбутнього додатку. На основі цих діаграм відбувається внутрішнє проектування програмного засобу.

3.2 Зовнішнє проектування

3.2.1 Мова програмування і середовище розробки

Розробка програми що здійснює часові заміри здійснюється з використанням мови програмування С. Вона підтримує декілька парадигм програмування: узагальнену та процедурну. Також вона має гарну швидкодію, масштабованість, підтримує модульне програмування, суть якого полягає в можливості роздільної компіляції і компоновки різних частин програми. Співробітник фірми Bell Labs Денис Рітчі створив мову С [30] в 1972 р під час спільної роботи з Кеном Томпсоном, як інструментальний засіб для реалізації операційної системи Unix [31], проте популярність цієї мови швидко переросла рамки конкретної операційної системи і конкретних завдань системного програмування. В даний час будь-яка інструментальна і операційна система не може вважатися повною, якщо до її складу не входить компілятор мови С. Рітчі не вигадував С просто з голови, прообразом послужила мову B, розроблена Томпсоном. Мова С була розроблена як інструмент для програмістів-практиків. Відповідно до цього головною метою його автора було створення зручного і корисної в усіх відношеннях мови.

Як і більшість імперативних мов, заснованих на традиції АЛГОЛ, С має можливості для структурного програмування і дозволяє здійснювати рекурсії, у той час, як система статичної типізації даних запобігає виникненню багатьох непередбачуваних операцій. У С увесь виконуваний код міститься у функціях. Параметри функції завжди передаються за значеннями. Передача

параметрів за вказівником реалізовується шляхом передачі значення вказівника. Гетерогенні сукупності типів даних (структури) дозволяють пов'язаним типам даних бути об'єднаними і маніпулювати ними, як єдиним цілим.

С також має такі специфічні властивості:

- змінні можуть бути прихованими у вкладених блоках
- слабка типізація; наприклад, символи можуть використовуватися, як цілі числа
- низькорівневий доступ до оперативної пам'яті шляхом перетворення машинних адрес на вказівники
- вказівники на функції і дані підтримують динамічний поліморфізм
- індексація масивів як вторинне поняття, визначається у термінах арифметики вказівників
- стандартизований препроцесор С для макроозначення, включення файлу з джерельним кодом, умовної трансляції тощо.
- відсутність вбудованих операторів вводу-виводу, потоків, обробки рядків і чисел з рухомою комою. Вся ця функціональність реалізується у бібліотеках виконання.
- відносно невелика кількість зарезервованих слів (32 у C89, і 37 у C99)
- Лексичні структури, які нагадують B більше за ALGOL, наприклад:
- `{ ... }` на відміну від ALGOL'івського `begin ... end`
- знак рівності для призначення (копіювання), як це робиться у мові Fortran
    - два знаки рівності використовуються для перевірки рівності (подібно до `.EQ.` у Fortran'і або одного знаку рівності у BASIC)
    - `&&` та `||` на відміну від ALGOL'івських `and` та `or` (цим вона семантично відрізняється від бітових операторів `&` та `|`).
    - велика кількість арифметичних і логічних операторів, на кшталт `+=`, `++`, `&=`,……

Мова С замислювалася як проміжна мова між мовами високого і низького рівня. Від компілятора намагалися домогтися продуктивності, близької до продуктивності Ассемблера, але в той же час зберегти можливість перенесення програм між комп'ютерними платформами, характерну для мов високого рівня. Хоча мова вимагає від програмістів високої дисципліни, вона не є суворою в формальних претензіях і допускає короткі формулювання. С - сучасна мова. Вона включає в себе ті керуючі конструкції, які рекомендовані теорією і практикою програмування. С – ефективна мова. Її структура дозволяє найкращим чином використовувати можливості сучасних персональних комп'ютерів. Програмування на цій мові відрізняється компактністю і швидкістю виконання. С - переносима і мобільна мова. С - потужна і гнучка мова. Велика частина операційної системи Unix, компілятори і інтерпретатори мов Фортран, Паскаль і Бейсік написані саме з її допомогою. С - зручна мова. Вона досить структурована, щоб підтримувати хороший стиль програмування і в той же час не пов'язана жорсткими обмеженнями. Надзвичайну популярність мова С отримала завдяки тому, що вона, як і Паскаль, є мовою структурного програмування, але дозволяє генерувати більш продуктивний і компактний робочий код. Характерним недоліком мови стала відносно висока складність вивчення в порівнянні з мовами Паскаль і Бейсік. Однак, у цієї мови є ще ряд недоліків. Для початку, в С досить високий поріг входження - дана мова буде важкою для вивчення новачкам. Так само, розроблена в середовищі хакерів, вона породжує аналогічний стиль програмування - небезпечний, стимулюючий написання заплутаного коду, як його ще називають у середовищі програмістів - write-only language. Синтаксис С справив значний вплив на появу інших мов, таких як Objective-C, C ++, Java і C #. Так, C ++ безпосередньо походить від С, але далі вони стали розвиватися незалежно один від одного. Цим можна пояснити їх деяку несумісність.

Програма була розроблена в QNX Momentics [32]. Комплект розробника QNX Momentics дозволяє прискорити розробку проекту на основі ОСРВ QNX

Neutrino на всіх його етапах - від вбудовування ОС на процессорную плату до інтеграції, аналізу та налагодження системи. Він включає в себе всі необхідні компоненти для розробки вбудованого ПО: інтуїтивно зрозуміле середовище розробки, вичерпний набір інструментів, пакети підтримки процесорних плат, кошти розробки драйверів - все це в одному інтегрованому комплекті. Також, завдяки підтримці різних мов програмування, різних інструментальних ОС і безлічі цільових процесорів, комплект QNX Momentics забезпечує розробників бути гнучкими, що дозволяє вибирати платформу і підхід до проектування, використовуючи один і той же звичний інструментарій.

Особливості та переваги QNX Momentics:

- Інтегроване середовище розробки на базі платформи Eclipse [33];
- Засоби розробки коду на C, C ++, Embedded C ++;
- Вбудовані засоби управління версіями;
- Інструментарій статичного і динамічного аналізу коду;
- Інструментарій аналізу ресурсів цільової системи;
- Підтримка оптимізують компіляторів GCC і ICC;
- Набір "майстрів", що автоматизують процес створення проекту;
- Візуальні засоби побудови багатомовних графічних додатків;
- Працює в середовищі Windows і Linux.

3.2.2 Методологія програмування

В інформатиці функціональне програмування - це парадигма програмування, де програми будуються за допомогою застосувань та складання функцій [34]. Це декларативна парадигма програмування, в якій визначеннями функцій є дерева виразів, кожна з яких повертає значення, а не послідовність імперативних висловлювань, що змінюють стан програми.

У функціональному програмуванні функції розглядаються як громадяни першого класу, що означає, що вони можуть бути прив'язані до імен (включаючи локальні ідентифікатори), передані як аргументи та повернуті з інших функцій,

як і будь-який інший тип даних. Це дозволяє писати програми в декларативному та складному стилі, де невеликі функції поєднуються модульним способом.

Функціональне програмування іноді трактується як синонім чисто функціонального програмування, підмножина функціонального програмування, яке розглядає всі функції як детерміновані математичні функції або чисті функції. Коли чиста функція викликається з деякими заданими аргументами, вона завжди повертає той самий результат, і на неї не може впливати будь-який змінний стан чи інші побічні ефекти. Це на відміну від нечистих процедур, поширених у імперативному програмуванні, які можуть мати побічні ефекти (наприклад, зміна стану програми або отримання вхідних даних від користувача). Прихильники чисто функціонального програмування стверджують, що, обмежуючи побічні ефекти, програми можуть мати менше помилок, їх легше налагоджувати та тестувати, а також більше підходити для офіційної перевірки.

Функціональне програмування сягає корінням в академічні кола, розвиваючись від лямбда-числення, формальної системи обчислень, заснованої лише на функціях. Функціональне програмування історично було менш популярним, ніж імперативне програмування, але багато функціональних мов сьогодні спостерігають використання у промисловості та освіті, зокрема Common Lisp, Scheme, Clojure, Wolfram Language, Рекет, Ерланг, OCaml, Хаскелл, і F #. Функціональне програмування також є ключовим для деяких мов, які досягли успіху в певних областях, наприклад, R [35] у статистиці, J, K та Q у фінансовому аналізі та XQuery / XSLT для XML [36]. Спеціальні декларативні мови, такі як SQL та Lex / Yacc, використовують деякі елементи функціонального програмування, наприклад, не дозволяють змінювати значення. Крім того, багато інших мов програмування підтримують програмування у функціональному стилі або реалізували функції функціонального програмування, такі як C , Kotlin [37], Perl [38], PHP [39], Python [40], Raku [41], та Скала [42].

Ітерація (цикл) у функціональних мовах зазвичай здійснюється за допомогою рекурсії. Рекурсивні функції викликаються самі, дозволяючи операції повторюватися, поки вона не досягне базового випадку. Як правило, рекурсія вимагає підтримання стека, який споживає простір у лінійній кількості до глибини рекурсії. Це може зробити рекурсію надмірно дорогою для використання замість обов'язкових циклів. Однак спеціальна форма рекурсії, відома як рекурсія хвоста, може бути розпізнана та оптимізована компілятором у тому самому коді, що використовується для реалізації ітерації на імперативних мовах. Оптимізація рекурсії хвоста може бути реалізована шляхом перетворення програми у стиль продовження передачі під час компіляції, серед інших підходів.

Стандарт мови C вимагає реалізації для підтримки належної рекурсії хвоста, тобто вони повинні дозволяти необмежену кількість активних викликів хвоста. Правильна рекурсія хвоста - це не просто оптимізація; це мовна функція, яка гарантує користувачам, що вони можуть використовувати рекурсію для вираження циклу, і це буде безпечно для простору. Більше того, на відміну від назви, він враховує всі виклики хвоста, а не лише рекурсію хвоста. Хоча правильна рекурсія хвоста зазвичай реалізується шляхом перетворення коду в обов'язкові цикли, реалізації можуть реалізувати його іншими способами. Наприклад, CHICKEN навмисно підтримує стек і дозволяє стеку переповнюватися. Однак, коли це трапляється, його збирач сміття вимагатиме простір назад, дозволяючи необмежену кількість активних викликів хвоста, навіть якщо це не перетворює рекурсію хвоста в цикл.

Поширені моделі рекурсії можна абстрагувати за допомогою функцій вищого порядку, причому катаморфізми та анаморфізми (або "складки" та "розгортання") є найбільш очевидними прикладами [43]. Такі схеми рекурсії відіграють роль, аналогічну вбудованим структурам управління, таким як цикли в імперативних мовах.

Більшість мов функціонального програмування загального призначення допускають необмежену рекурсію і є повнотою Тьюрінга, що робить проблему

зупинки невирішуваною, може спричинити необгрунтованість рівномірних міркувань і, як правило, вимагає внесення невідповідності в логіку, виражену системою типів мови. Деякі мови спеціального призначення, такі як Coq, дозволяють лише обгрунтовану рекурсію і суворо нормалізуються (необмежені обчислення можуть бути виражені лише нескінченними потоками значень, що називаються кодами). Як наслідок, ці мови не можуть бути повноцінними за Тьюрінгом, і вираження в них певних функцій неможливо, але вони все одно можуть виражати широкий клас цікавих обчислень, уникаючи проблем, викликаних необмеженою рекурсією. Функціональне програмування, обмежене обґрунтованою рекурсією з кількома іншими обмеженнями, називається загальним функціональним програмуванням.

3.2.3 Вибір парадигми програмування

Парадигма проектування характеризує основні принципи програмування, а саме, точку зору програміста на роботу продукту. Парадигма формулює сукупність ідей і понять, що визначає стиль розробки програми.

В результаті аналізу постановки задачі було вирішено застосовувати функціональний метод проектування. Цей вибір обумовлений наступними його властивостями:

– Функціональна (процедурна) модель дозволяє повною мірою використовувати виразні можливості об’єктних мов, таких як C, Ada [44] та ін., один із яких передбачалося взяти як мову розробки;

– використання функціонального (процедурного) підходу істотно підвищує рівень уніфікації розробки і придатність для повторного використання не тільки програм, але і проектів;

– використання функціональної моделі приводить до побудови систем на основі стабільних проміжних описів, що спрощує процес внесення змін. Це дає системі можливість розвиватися поступово і не

приводить до повної її переробки навіть у випадку істотних змін вихідних вимог;

– функціональний підхід складається з ряду добре продуманих етапів проектування, що зменшує ступінь ризику і підвищує впевненість у правильності прийнятих рішень;

– функціональна модель інтуїтивно зрозуміла для розробників різного рівня підготовки.

Таким чином, функціональний (процедурний) підхід якнайкраще задовольняє потреби проектування даного програмного продукту.

3.3 Розробка функціональної моделі

3.3.1 Опис функцій та змінних кожного програмного модуля

Опис відповідальності змінних та функцій кожного модуля буде наведено на рисунках 1.13 – 1.27.

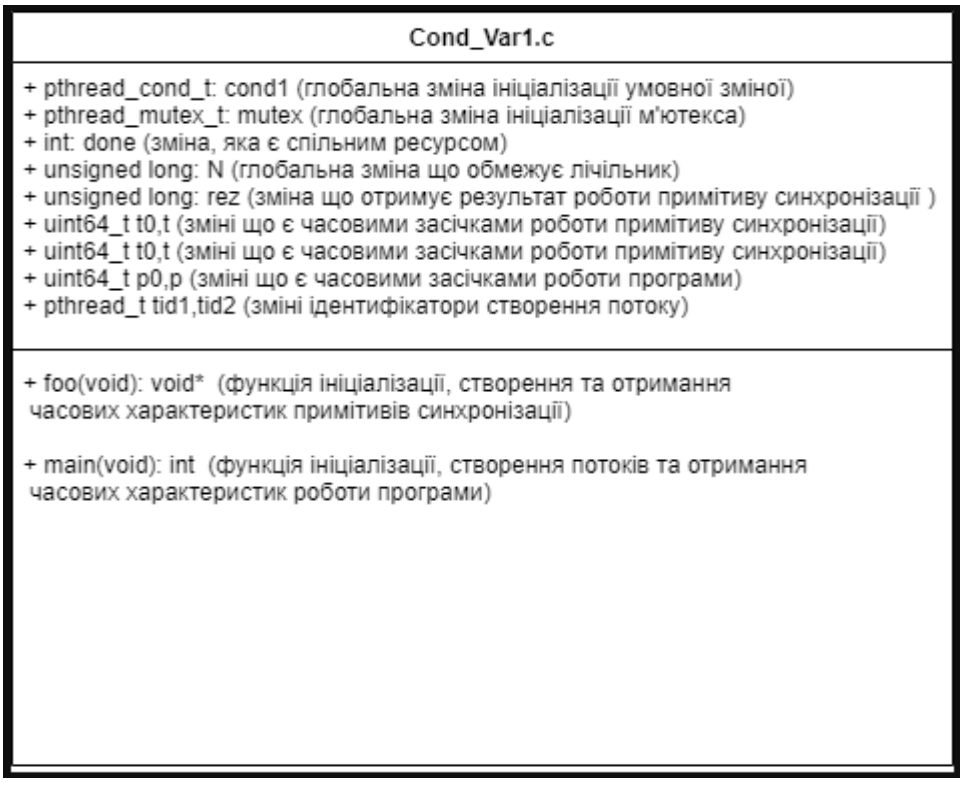


Рисунок 1.13 – Опис модуля Cond_Var1.c

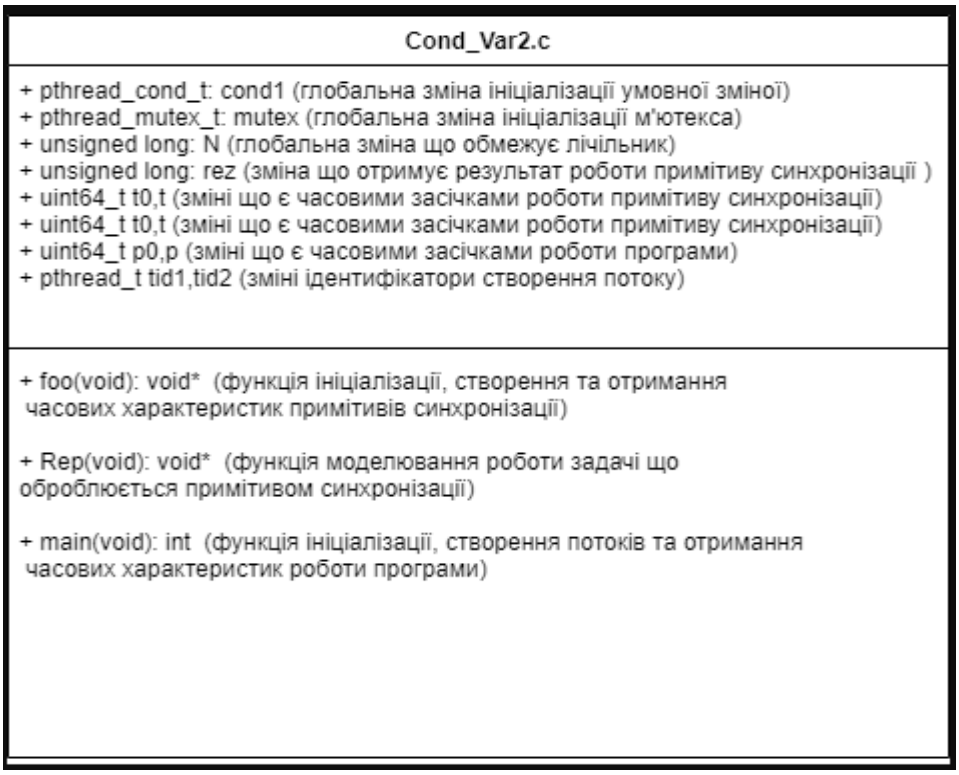


Рисунок 1.14 – Опис модуля Cond_Var2.c

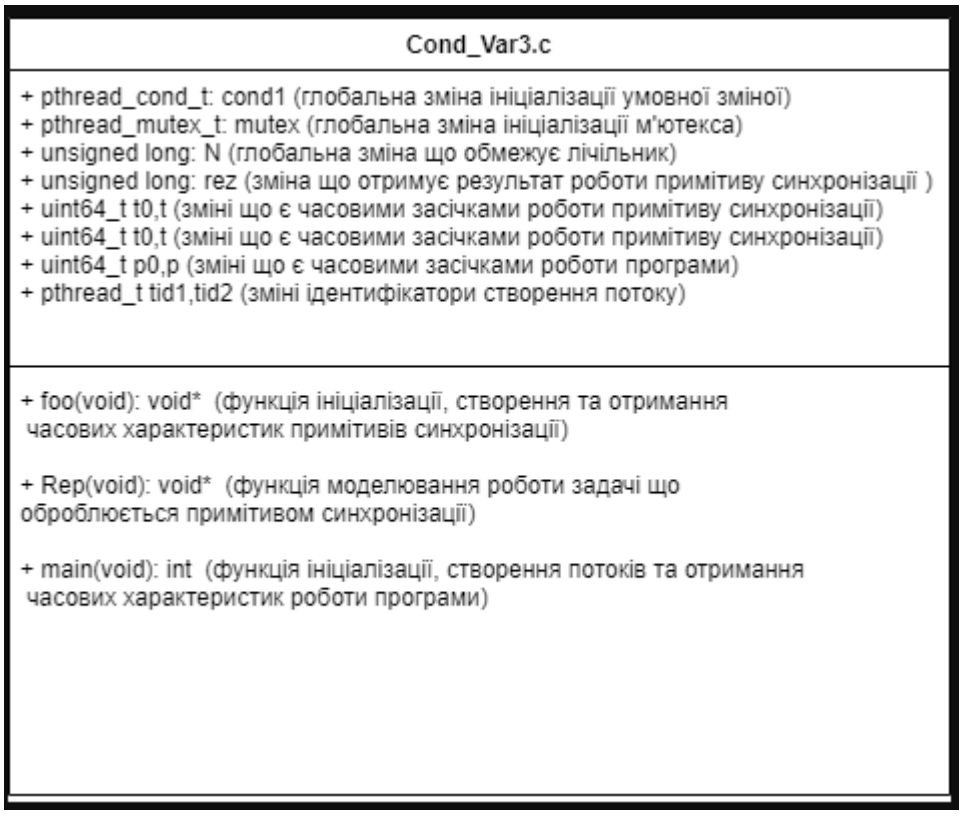


Рисунок 1.15 – Опис модуля Cond_Var3.c

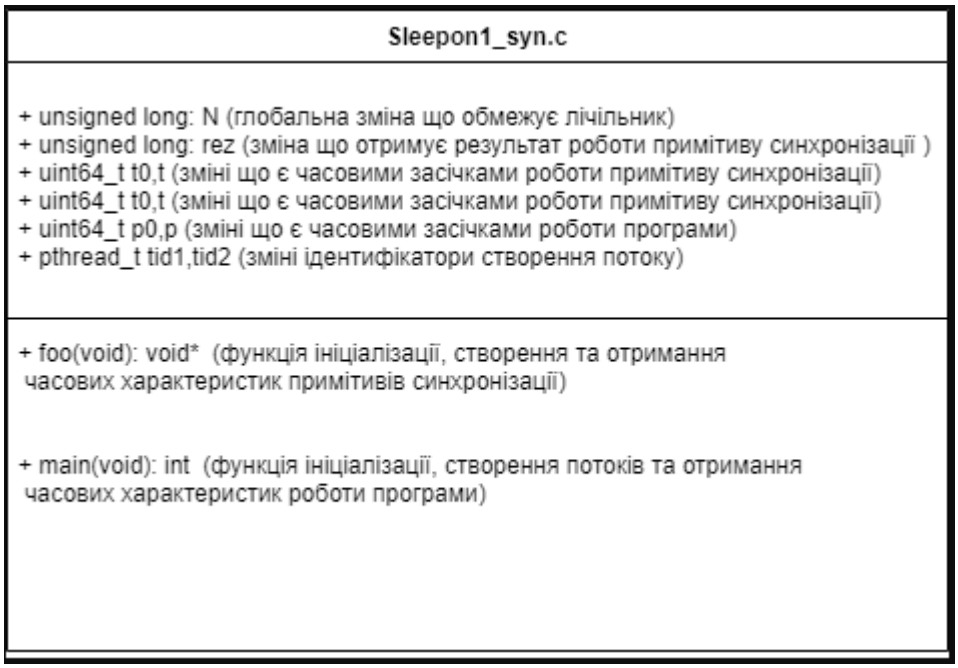


Рисунок 1.16 – Опис модуля Sleepon1_syn.c

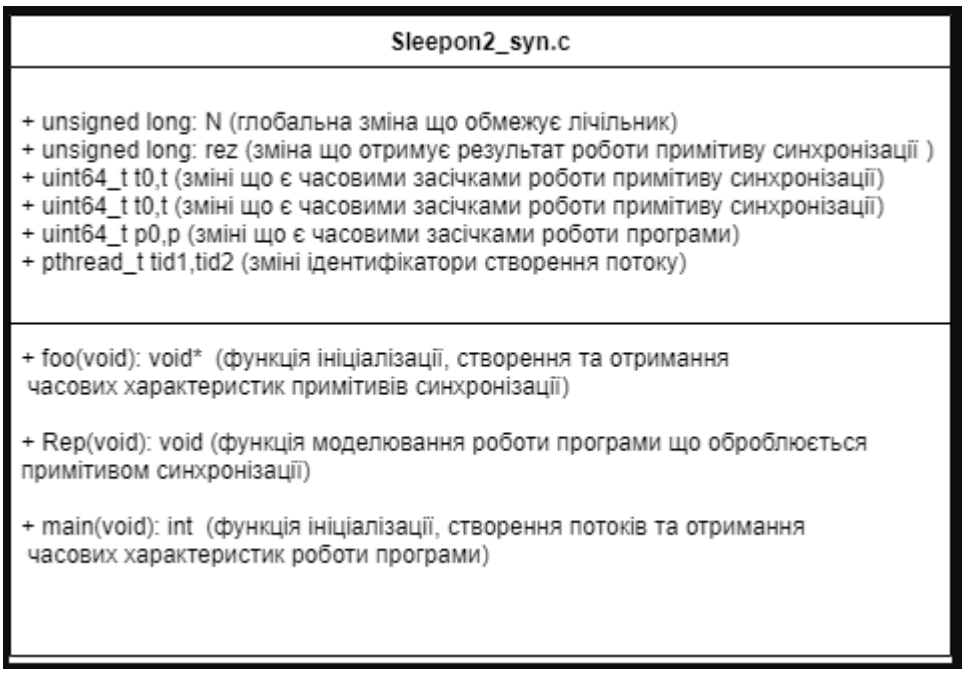


Рисунок 1.17 – Опис модуля Sleepon2_syn.c

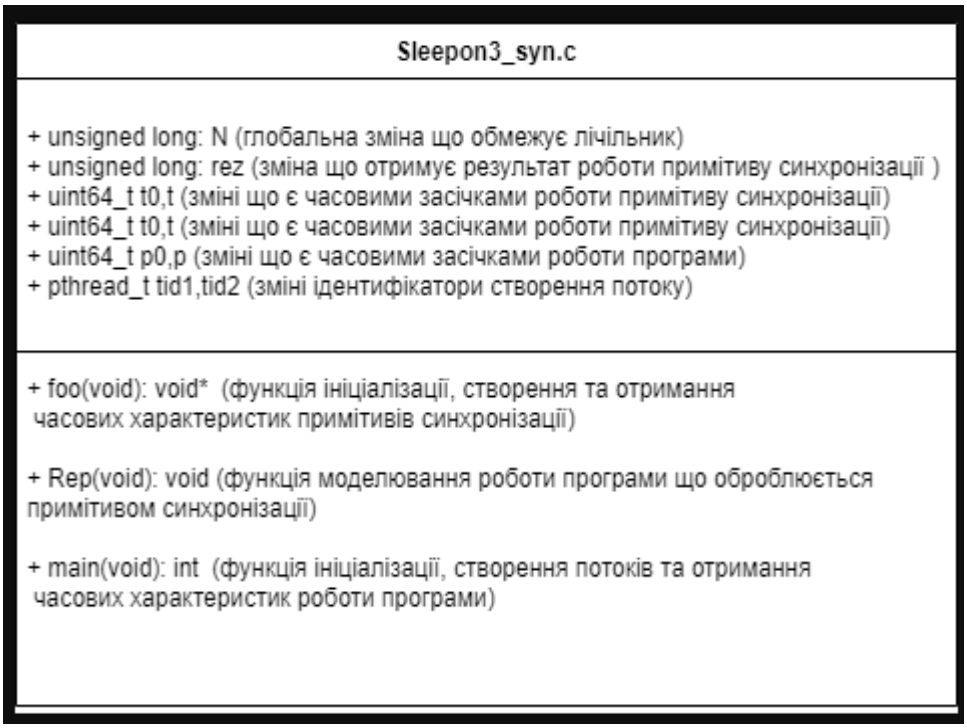


Рисунок 1.18 – Опис модуля Sleepon3_syn.c

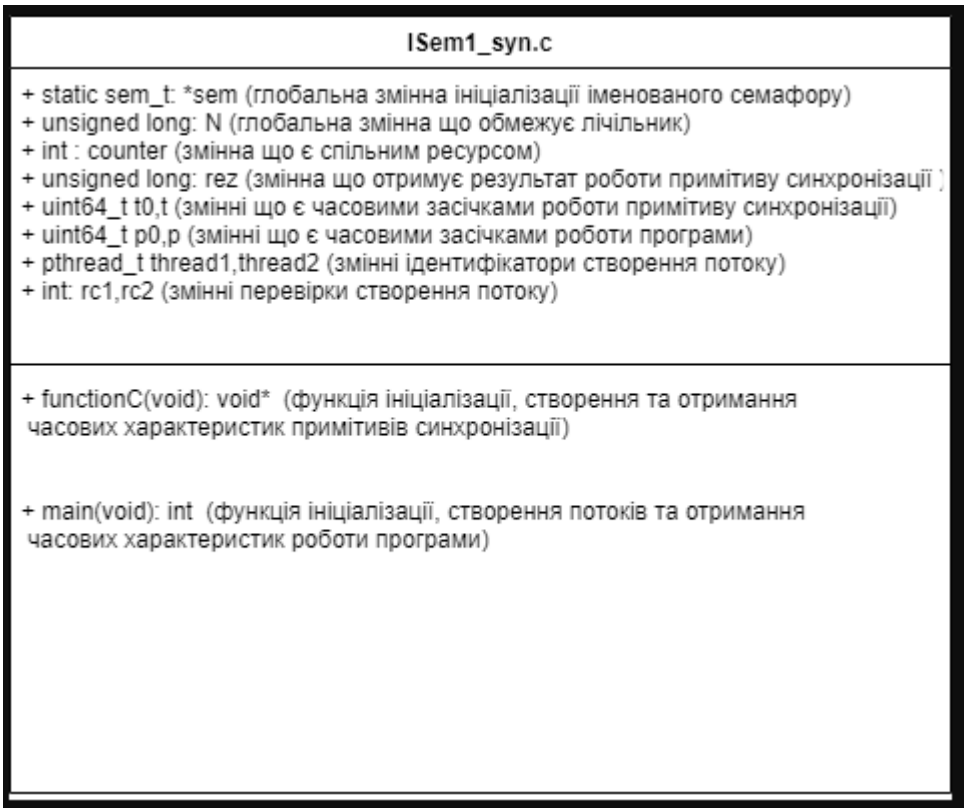


Рисунок 1.19 – Опис модуля ISem1_syn.c

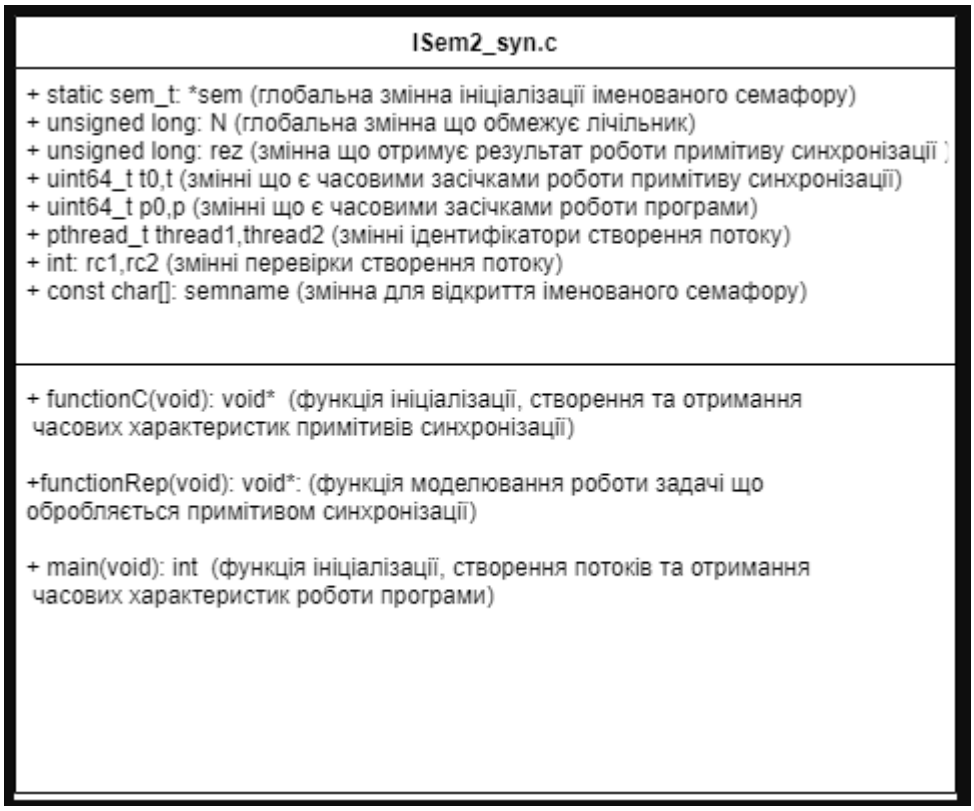


Рисунок 1.20 – Опис модуля ISem2_syn.c

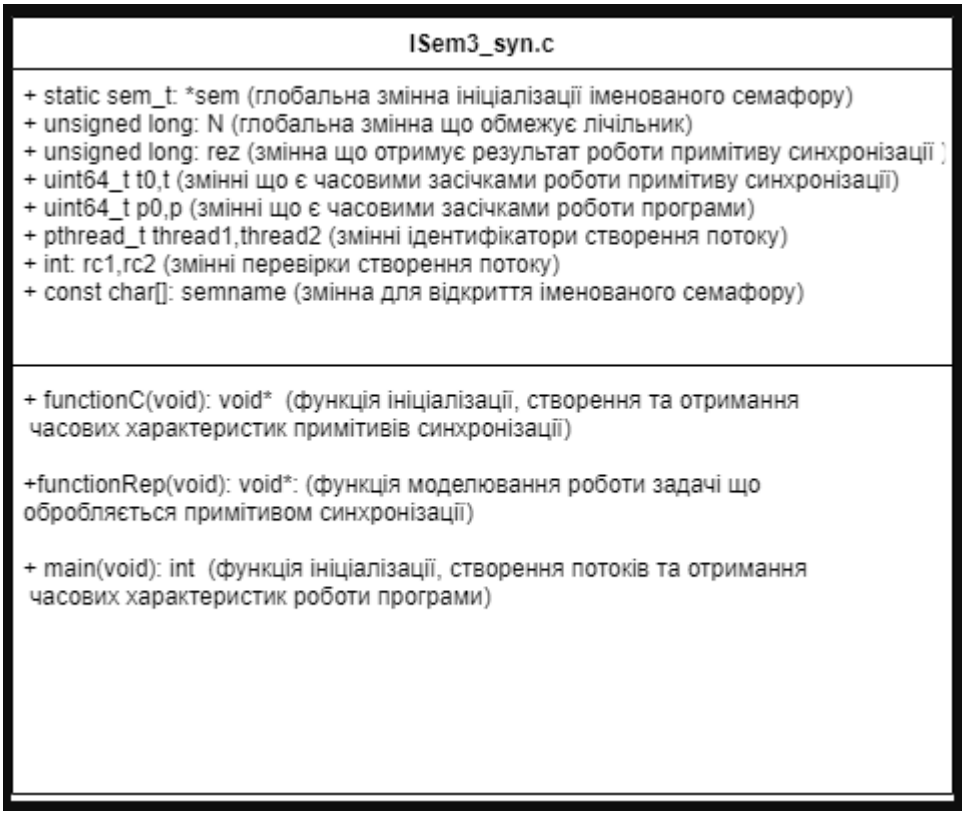


Рисунок 1.21 – Опис модуля ISem3_syn.c

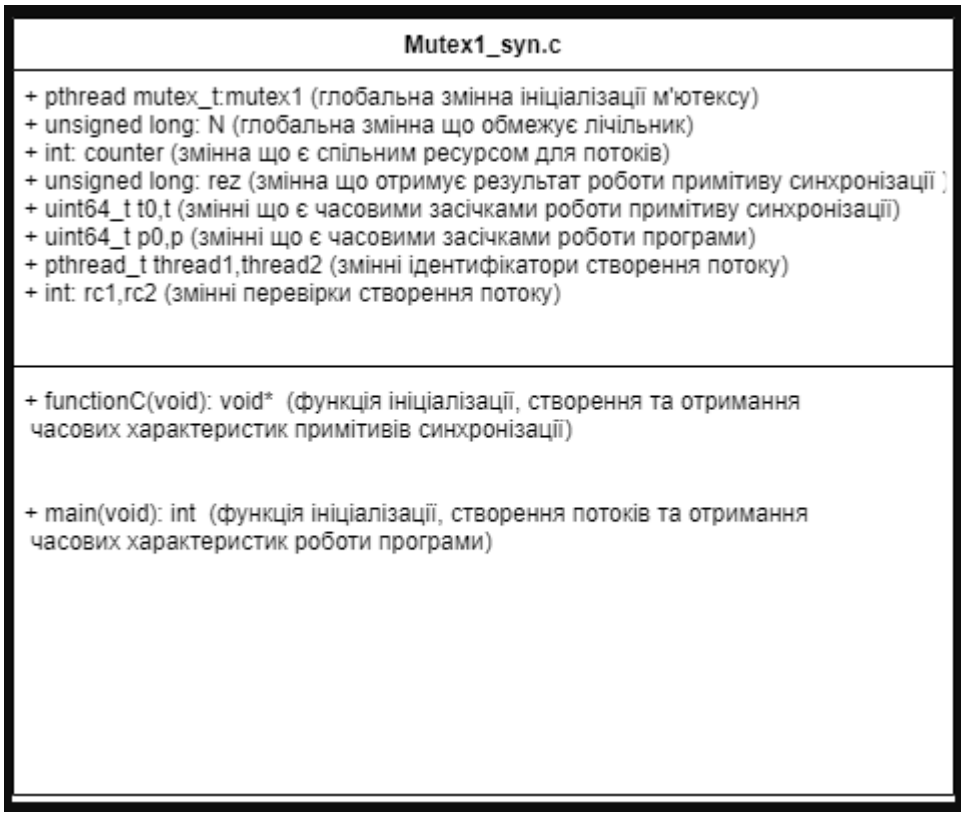


Рисунок 1.22 – Опис модуля Mutex1_syn.c

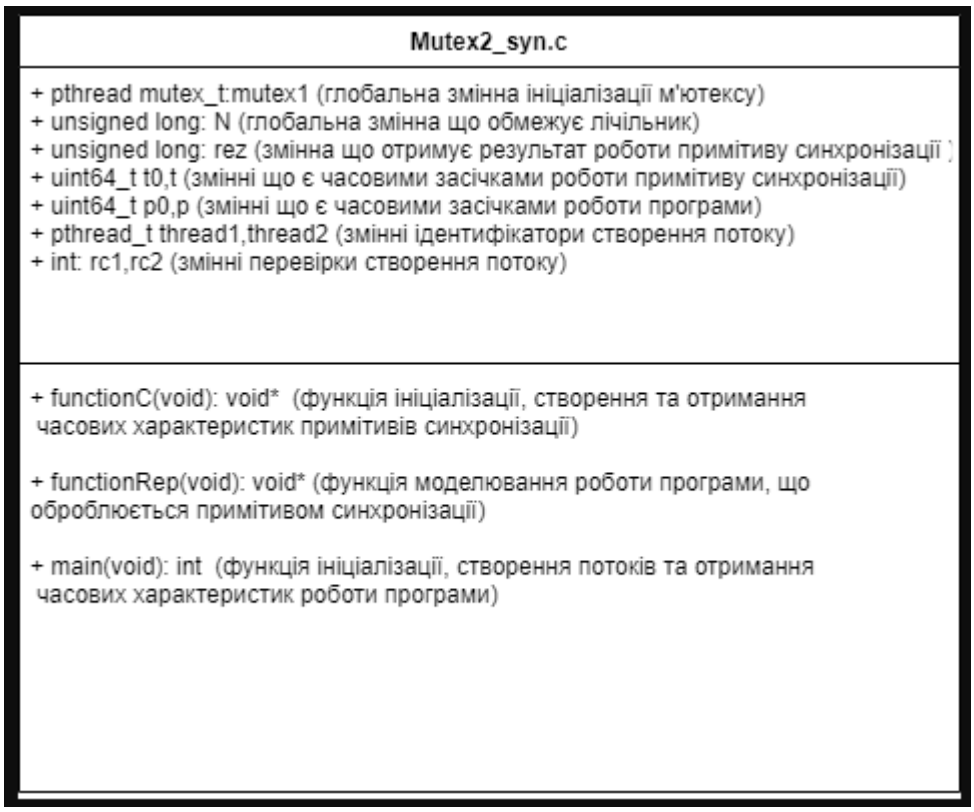


Рисунок 1.23 – Опис модуля Mutex2_syn.c

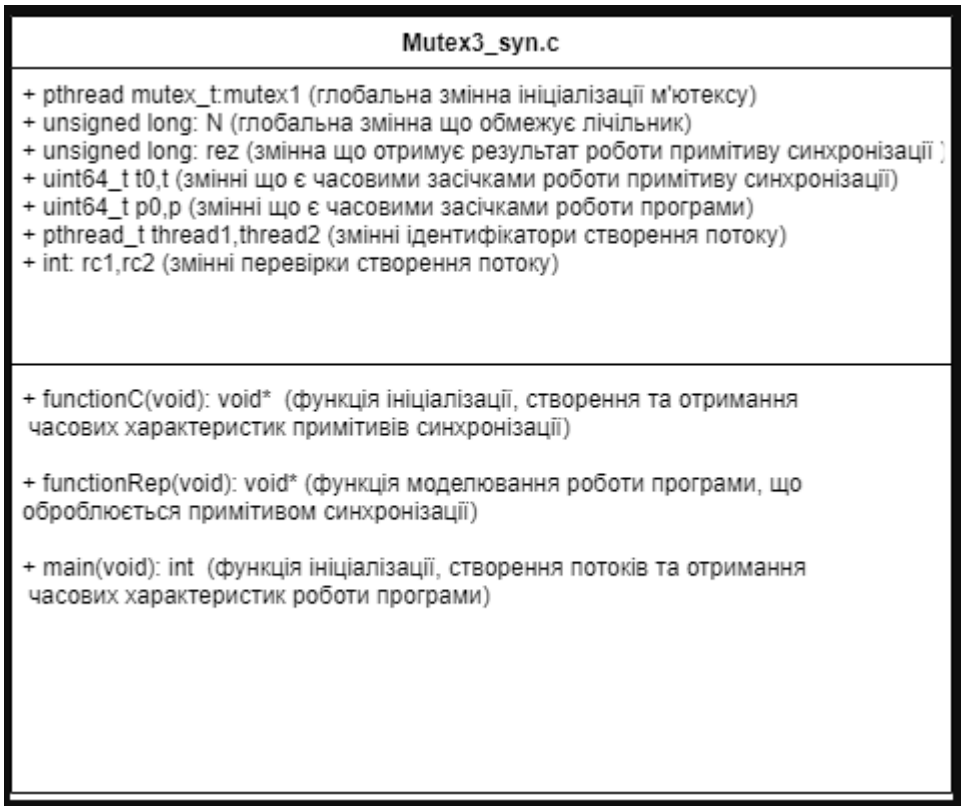


Рисунок 1.24 – Опис модуля Mutex3_syn.c

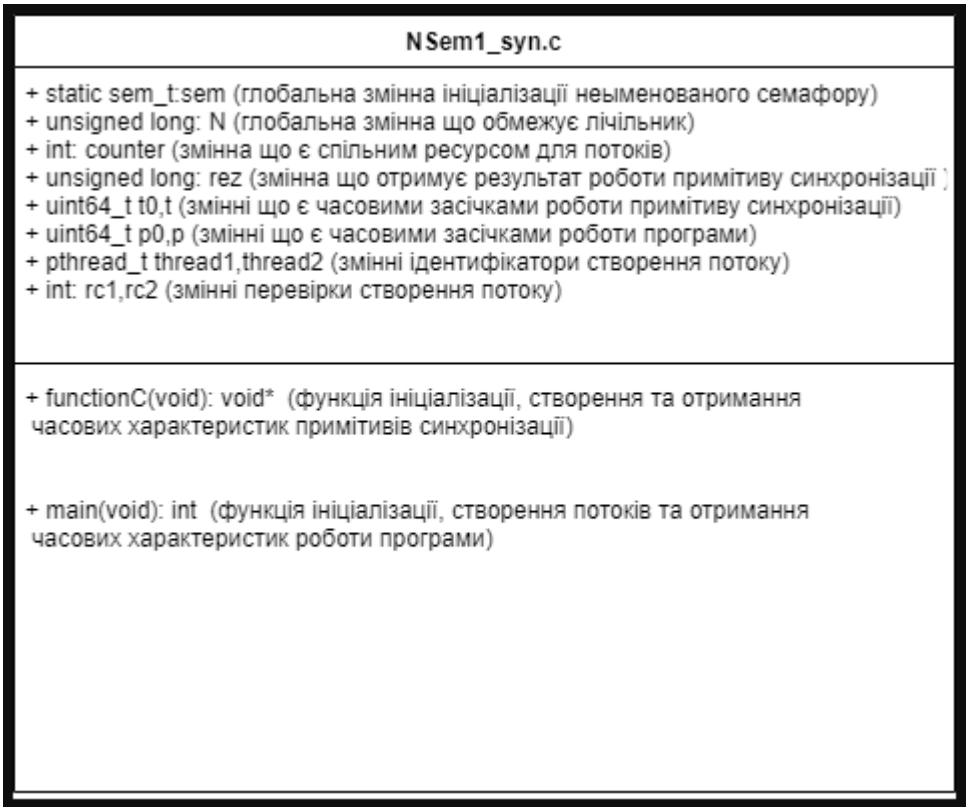


Рисунок 1.25 – Опис модуля NSem1_syn.c

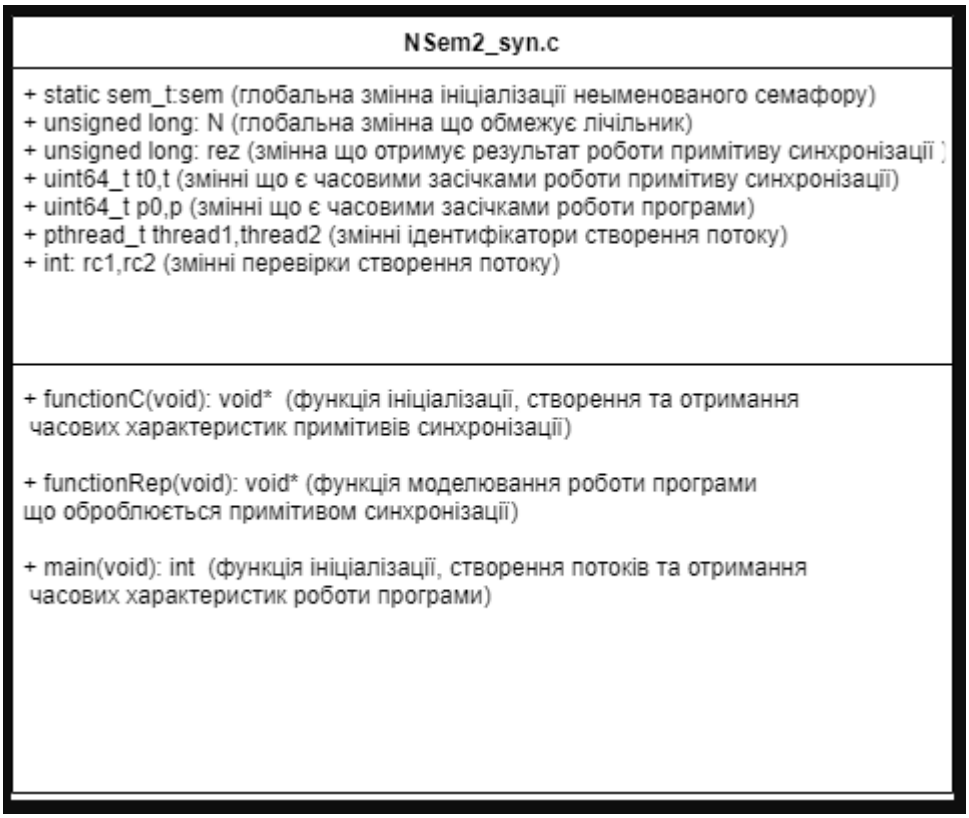


Рисунок 1.26 – Опис модуля NSem2_syn.c

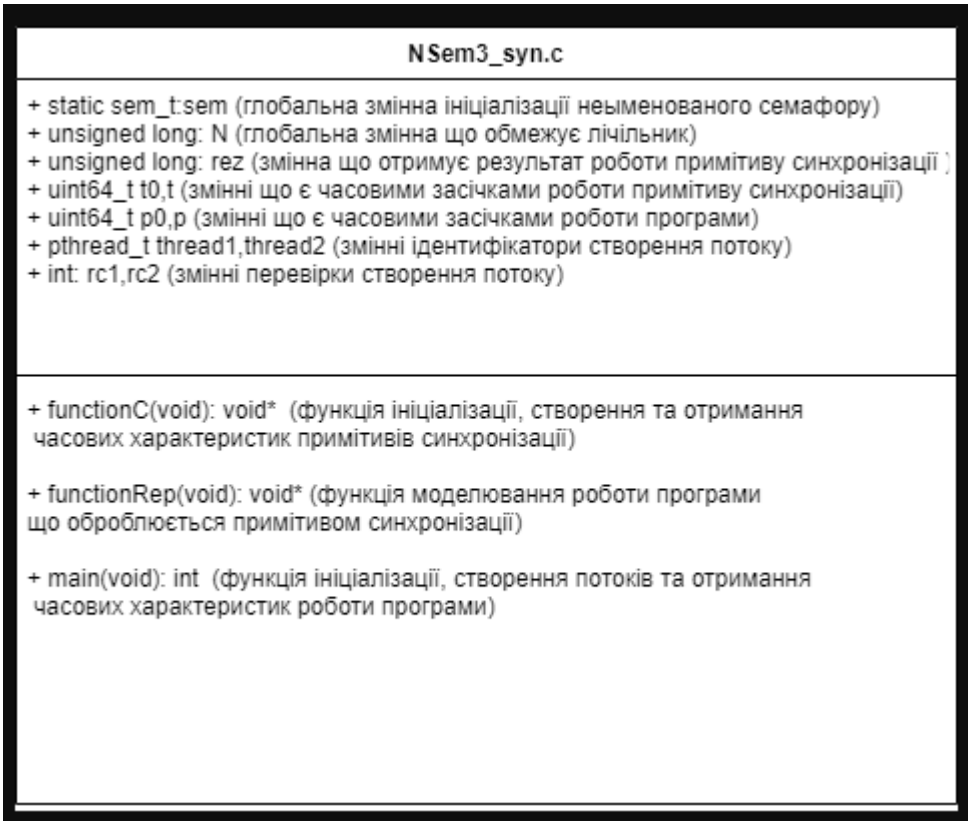


Рисунок 1.27 – Опис модуля NSem3_syn.c

3.4 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача є важливим етапом створення програмного продукту. Незалежно від методів реалізації користувач сприймає програму та взаємодіє з нею саме через інтерфейс. Інтерфейс програми має бути зручним та інтуїтивно зрозумілим. У разі виникнення непередбачуваних станів, користувач повинен отримати вичерпне повідомлення про помилку, можливі причині її виникнення та рекомендовані подальші дії. Після переходу до папки «QNX C++ SyncPrimComp» з'являється головне вікно програми. На рисунку 1.28 зображено інтерфейс, що надається терміналом операційної системи.

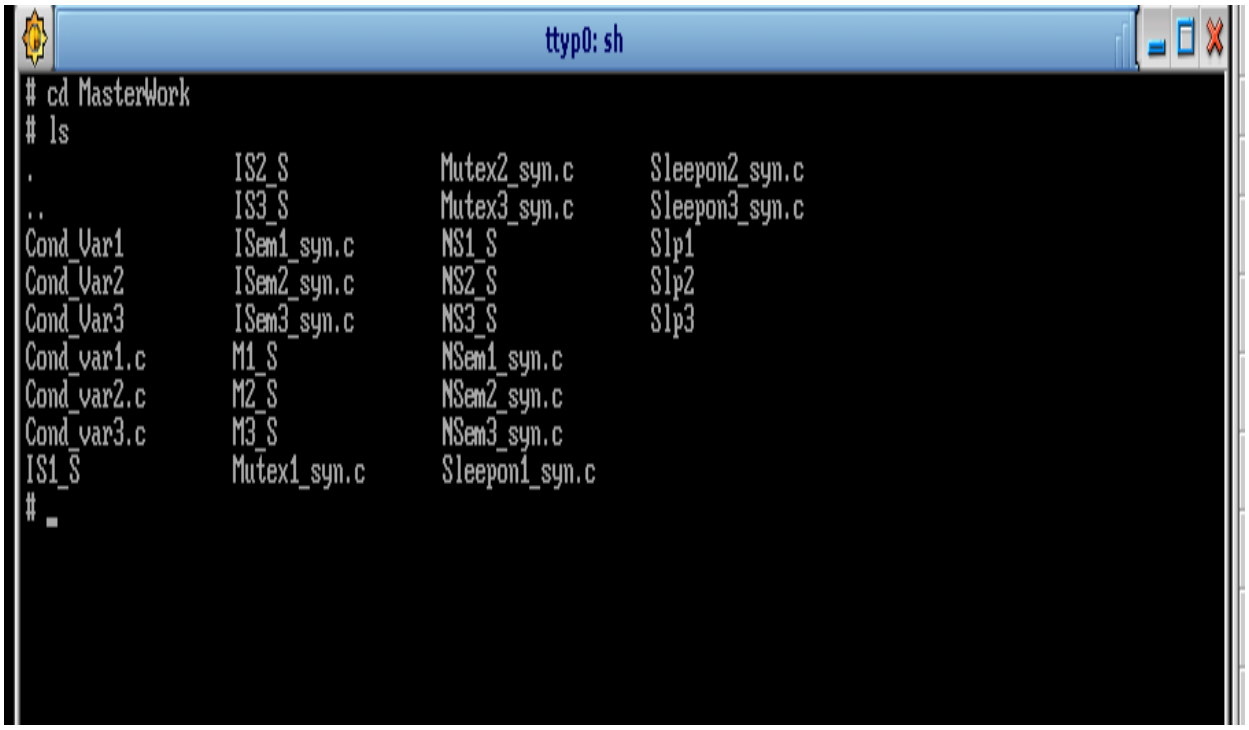


Рисунок 1.28 – Інтерфейс програми

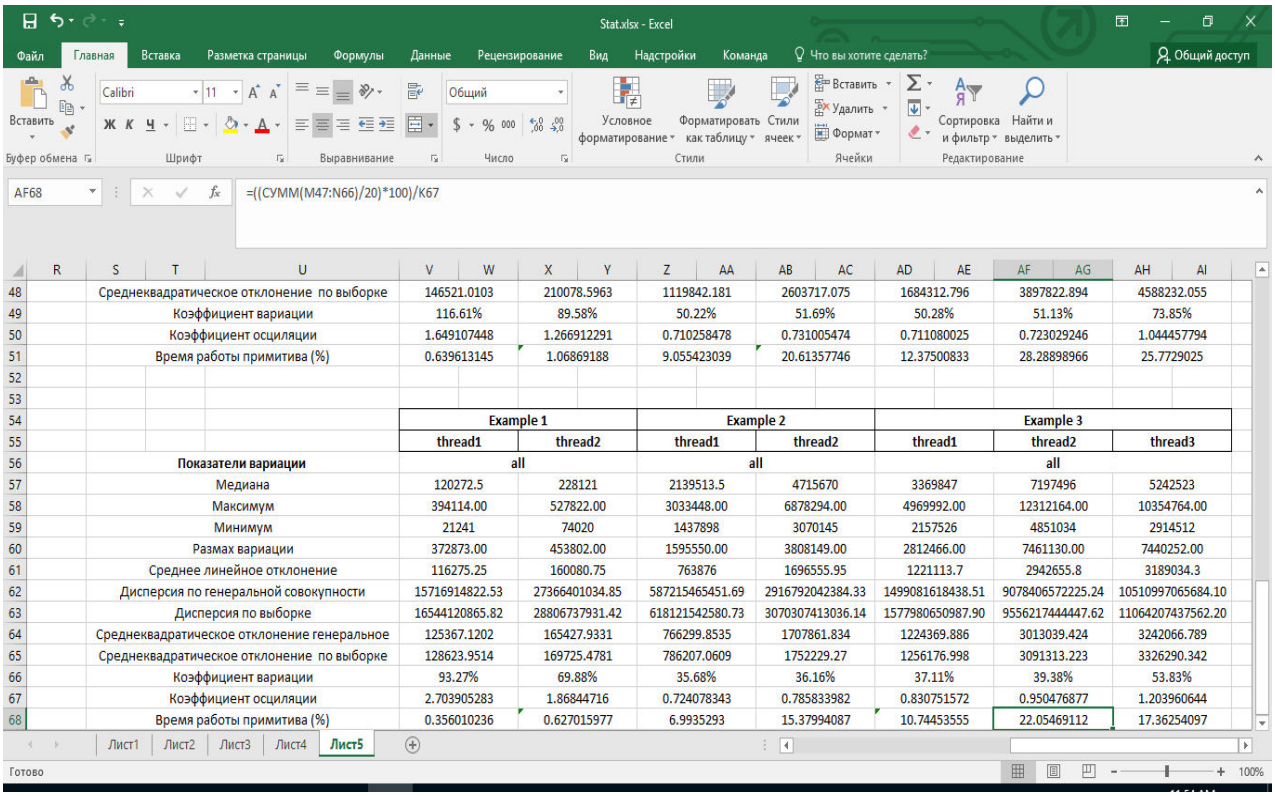
Головне меню програми складається з наступних пунктів що є базовими для терміналу операційної системи:

– «Зменшити вікно» (відповідає за зменшення);

– «Збільшити вікно» (відповідає за збільшення вікна);

– «Закрити вікно» (відповідає за закриття вікна).

Потім отримані дані часової характеристики роботи програм переносяться у відповідні поля програми Excel 2016, де здійснюється їх статистична обробка та візуалізація. Інтерфейс цієї частини програми зображено на рисунку 1.29.



		Example 1		Example 2		Example 3		
		thread1	thread2	thread1	thread2	thread1	thread2	thread3
48	Среднеквадратическое отклонение по выборке	146521.0103	210078.5963	1119842.181	2603717.075	1684312.796	3897822.894	4588232.055
49	Коэффициент вариации	116.61%	89.58%	50.22%	51.69%	50.28%	51.13%	73.85%
50	Коэффициент осциляции	1.649107448	1.266912291	0.710258478	0.731005474	0.711080025	0.723029246	1.044457794
51	Время работы примитива (%)	0.639613145	1.06869188	9.055423039	20.61357746	12.37500833	28.28898966	25.7729025
52								
53								
54								
55								
56	**Показатели вариации**	**all**		**all**		**all**		
57	Медиана	120272.5	228121	2139513.5	4715670	3369847	7197496	5242523
58	Максимум	394114.00	527822.00	3033448.00	6878294.00	4969992.00	12312164.00	10354764.00
59	Минимум	21241	74020	1437898	3070145	2157526	4851034	2914512
60	Размах вариации	372873.00	453802.00	1595550.00	3808149.00	2812466.00	7461130.00	7440252.00
61	Среднее линейное отклонение	116275.25	160080.75	763876	1696555.95	1221113.7	2942655.8	3189034.3
62	Дисперсия по генеральной совокупности	15716914822.53	27366401034.85	587215465451.69	2916792042384.33	1499081618438.51	9078406572225.24	10510997065684.10
63	Дисперсия по выборке	16544120865.82	28806737931.42	618121542580.73	3070307413036.14	1577980650987.90	9556217444447.62	11064207437562.20
64	Среднеквадратическое отклонение генеральное	125367.1202	165427.9331	766299.8535	1707861.834	1224369.886	3013039.424	3242066.789
65	Среднеквадратическое отклонение по выборке	128623.9514	169725.4781	786207.0609	1752229.27	1256176.998	3091313.223	3326290.342
66	Коэффициент вариации	93.27%	69.88%	35.68%	36.16%	37.11%	39.38%	53.83%
67	Коэффициент осциляции	2.703905283	1.86844716	0.724078343	0.785833982	0.830751572	0.950476877	1.203960644
68	Время работы примитива (%)	0.356010236	0.627015977	6.9935293	15.37994087	10.74453555	22.05469112	17.36254097

Рисунок 1.29 – Інтерфейс обробки і візуалізації результатів програми

Головне меню програми складається з наступних пунктів що є базовими для інтерфейсної частини програми Excel 2016. Детальний опис інтерфейсу програми буде доволі об’ємним тому з ним можна ознайомитися за наступним посиланням.

3.5 Вибір стратегії тестування

Тестування програмного забезпечення охоплює цілий ряд видів діяльності, аналогічний послідовності процесів розробки програмного забезпечення. Сюди входять постановка задачі для тесту, проектування, написання тестів, тестування тестів і, нарешті, виконання тестів і вивчення результатів тестування. Вирішальну роль грає проектування тесту. Можливий цілий спектр підходів до вироблення стратегії проектування тестів. Для досягнення максимальної якості

тестування під кожен метод або модуль програми, що тестується необхідно обрати найбільш вдалий метод або набір методів, що забезпечать необхідний результат. При цьому необхідно пі-дібрати найкраще співвідношення між часом, що буде витрачено на тестування, та якістю тестування. Набір тестів повинен мати мінімальну збитковість, але при цьому максимально охоплювати функціональність системи

Для ефективного тестування були використані методи «білого ящика» [45] та «чорного ящика» [46].

Для функцій зі складними умовами був використаний метод тестування «білим ящиком» – метод покриття умов та рішень [47]. Даний метод дозволяє виявити непрохідні гілки умов, виявити оператори, які ніколи не будуть виконуватись. Метод покриття умов та рішень дозволяє виявити найбільшу кількість помилок серед методів «білого ящика». Даний метод полягає у записі числа тестів достатнього для того, щоб були покриті всі оператори, умови та рішення. Кожний оператор, умова та рішення повинні бути виконані хоча б один раз. Якщо після зіставлення тестів залишаються не покриті оператори, умови чи рішення, то потрібно доповнити свій набір тестів.

Функції, в яких немає складних умов, тестувались методом покриття умов, так як цього достатньо для виявлення помилки.

Метод тестування «чорним ящиком» припускає розробку тестів на основі аналізу специфікації програми. Тестування програми обмежується використанням невеликої підмножини всіх можливих вхідних даних. Правильно обраний тест цієї підмножини повинен володіти двома властивостями:

- зменшувати число тестів, які повинні бути розроблені для досягнення заздалегідь визначеної мети тестування;
- покривати значну частину інших можливих тестів, що в деякому ступені свідчать про наявність або відсутності помилок до і після застосування цієї обмеженої підмножини значень вхідних даних.

Серед методів тестування «чорним ящиком» було застосовано метод припущення про помилку.

3.6 Тестування програми

3.6.1 Тестування функції foo

Призначенням функції foo є отримання часових характеристик примітиву синхронізації за допомогою спеціальної функції ClockCycles(), що отримує часові характеристики безпосередньо підлаштовуючись під тактову частоту процесора. Отримані результати цієї функції є доволі точими і вимірюються в тіках системного процесора та можуть бути конвертовані в наносекунди.

Вхідні дані від користувача функція не отримує.

Вихідними даними функції є часові характеристики примітиву синхронізації.

Текст методу:

```
void* foo()
{
unsigned long i;
unsigned long rez=0;
uint64_t t=0, t1;
while(i++!N)
{
t1=ClockCycles();
pthread_mutex_lock(&mutex);
if(done==1) {
done++;
```

```
functionRep();
printf(“Done value %d\n”, done);
printf(“Waiting on conditional variable cond1\n”);
pthread_cond_signal(&cond1);
}
else
{
printf(“Signaling conditional variable cond1\n”);
pthread_cond_signal(&cond1);
}
pthread_mutex_unlock(&mutex);
t+=ClockCycles()-t1;
sched_yield();
rez=t/N;
printf(“Returning thread\n”);
printf(“%i cycles %i on cond var “, pthread_self(), t);
printf(“%i\n”, rez);
}
return NULL;
}
```

Тестування даного методу було вирішено проводити одним із методів білого ящика, оскільки дані методи дозволять досягти кращої якості тестування, ніж методи чорного ящика і не є занадто складними для використання при

тестуванні даного методу. Серед методів білого ящика було обрано метод покриття умов. Даний метод потребує перевірки всіх можливих результатів умов, тому під час виконання всіх тестів кожна умова повинна хоча б один раз виконатись та один раз не виконатись.

Спочатку було створено таблицю умов (табл. 3.1).

Таблиця 3.1 – Умови функції foo

№	Умова
1	i++!N
2	done==1
3	done!=1

Перелік розроблених тестів зображено в табл. 3.2.

Таблиця 3.2 – Розроблені тести для функції foo

Номер тесту	Вхід	Вихід

1	i=1 N=3	Returning thread 123456 cycles 30000 on cond var 10000; 345678 cycles 300000 on cond var 100000; 43678065 cycles 3000000 on cond var 1000000;
2	done=1	Returning thread 451289 cycles 300 on cond var 100;
3	done =3	Waiting for signaling conditional variables Returning thread 986756 cycles 600 on cond var 200;

За результатами тестування всі тести були виконані успішно та всі можливі умови були щонайменше один раз виконані та один раз не виконані (табл. 3.3).

Таблиця 3.3 – Покриття умов тестами

Номер тесту	Умови		
	1	2	3
1	-	-	-
2	+	+	-
3	+	-	+

3.7 Відлагодження програми

Під час відлагодження програми виникла помилка з оператором «exit» при отримані часових характеристик роботи примітиву синхронізації м’ютекс в програмному модулі Mutex1_syn.c. Помилка виникла у наступному фрагменты коду:

```
int main()
{
int rc1,rc2;
pthread_t thread1,thread2;
uint64_t p=0,p1;
p1=ClockCycles();
if((rc1=pthread_create( &thread1,NULL,&functionC,NULL)))
{
printf(“Thread creation failed: %d\n”,rc1);
}
if((rc2=pthread_create( &thread2,NULL,&functionC,NULL)))
{
printf(“Thread creation failed: %d\n”,rc2);
}

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
```

```
p+=ClockCycles()-p1;
printf("Program time: %i\n",p);
exit(EXIT_SUCCESS)
}
```

Компілятор проінформував про помилку під час компіляції програми що наведено на рисунку 1.30.

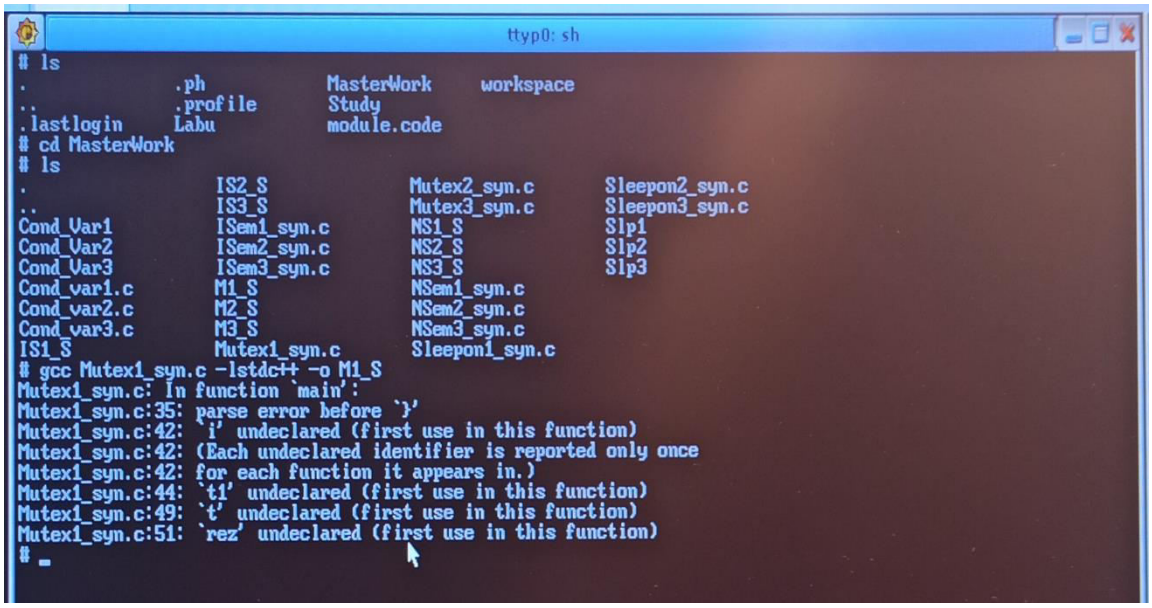


Рисунок 1.30 Повідомлення про помилку

Для того щоб виправити помилку було запущено середовище розробки програм QNX Momentics IDE. Була встановлена контрольна точка, в фрагменті коду який підлягав перевірці. Методом проб та помилок було знайдено та виправлено непрацюючий фрагмент. Процес пошуку помилки наведено на рисунку 1.31. Успіший запуск програми після відладки наведено на рисунку 1.32.

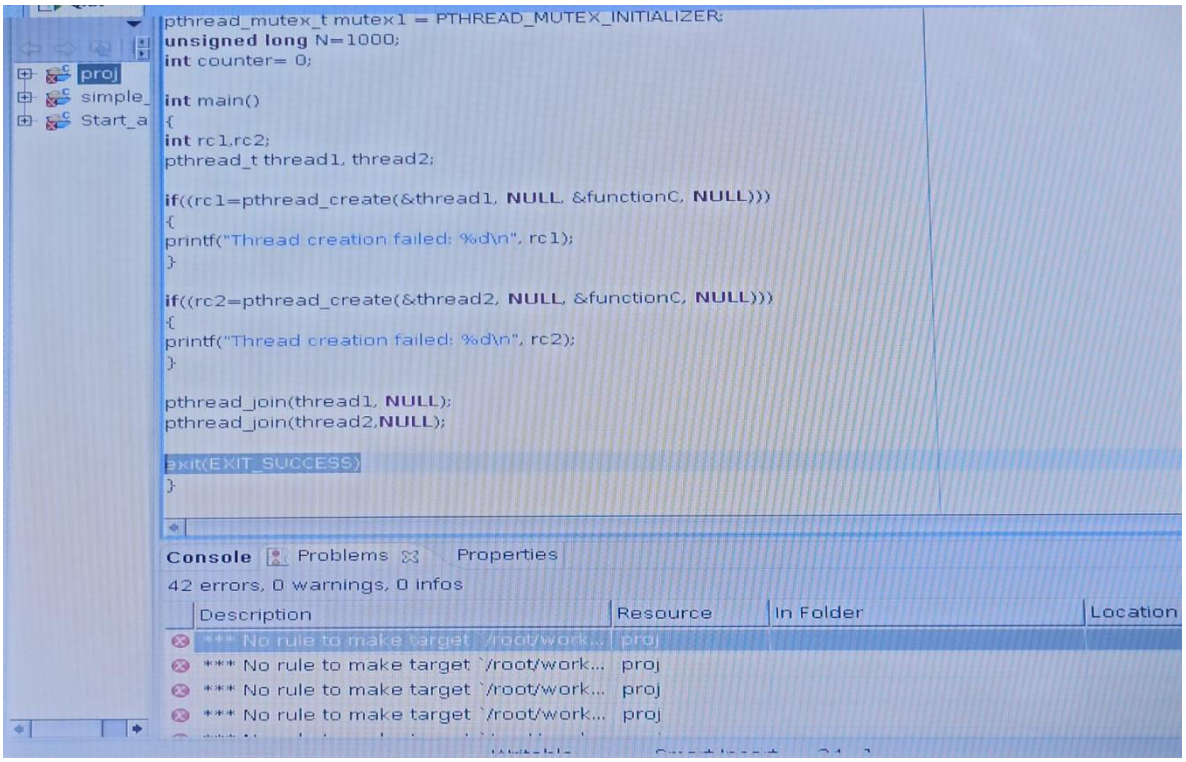


Рисунок 1.31 Відладлагодження фрагменту програми

Успіший запуск програми після відладки наведено на рисунку 1.32.

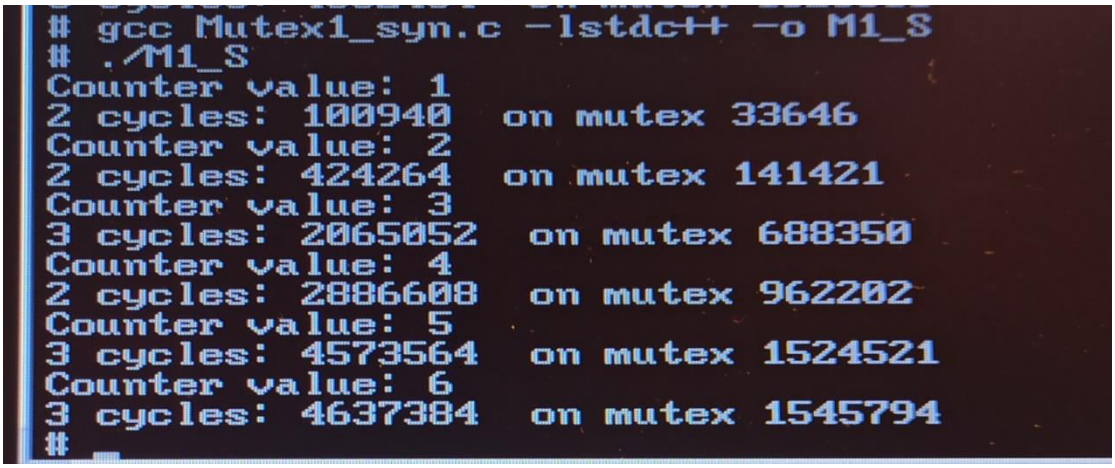


Рисунок 1.32 Успішний запуск програми після відлагодження

Функціональна частина програми налагоджувалась засобами QNX Momentics IDE, а інтерфейсна засобами терміналу операційної системи.

4 АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Методи дослідження

Для того щоб отримати часові характеристики роботи примітиву синхронізації та всієї програми було використано спеціальну програмну функцію ClockCycles(), що дозволяє отримати бажаний час роботи в системніх тіках процесора, що за баженням може бути переведено в час в наносекундах. Було проведено 20 успішних запусків програми для групи два та 30 успішних запусків для групи один. За результатами була набрана статистика роботи програми та примітивів синхронізації для кожної з задач.

Примітиви для порівняння між собою були розбиті на дві групи:

- умовна змінна та сліпон;
- іменований та неіменований семафор та мютекс.

Модель першої задачі це боротьба примітивів синхронізації за спільний ресурс який представляє собою змінну яку вони мають інкрементувати. Кількість потоків в першій задачі дорівнює двум.

Модель другої задачі це боротьба примітивів синхронізації за спільний ресурс який представляє собою цикл на 100000 ітерацій. Кількість потоків в другій задачі дорівнює двум.

Модель третьої задачі це боротьба примітивів синхронізації за спільний ресурс який представляє собою цикл на 100000 ітерацій. Кількість потоків в третій задачі дорівнює трьом.

4.2 Збір даних для аналізу

В результаті часових замірів були отримані дані наведені в таблицях 1.1-1.5.

Таблиця 1.1 Результати часових замірів умовної змінної

Задача 1		**Задача 2**		**Задача 3**		
thread1	**thread2**	**thread1**	**thread2**	**thread1**	**thread2**	**thread3**
47764	735844	1326449	2986160	127597828	136712777	112901868
120129	769254	1368890	3156965	128890090	138009565	114353085
148661	1267512	1400157	3189630	130261122	139304936	115641333
33816	430276	1334494	2984080	128532401	137979408	113838753
63834	630741	1376657	3155448	129829346	139276972	115126044
91730	660025	1407262	3184678	131116185	140567492	116436474
34224	435218	1318370	2974049	129602517	139109194	114908506
64030	636716	1360576	3172768	130896732	140396498	116204681
92593	678269	1391208	3201152	132186172	141684248	117495270
47140	1259246	706686	2395544	130228125	139794753	115527274
215994	1290461	748757	2595805	131514600	141064645	116828480
244642	2237684	778890	2624334	132804674	142370294	118140230
47046	1236236	704082	2395170	127021236	136857505	112885418
198754	1266161	746017	2595161	128455930	138233704	114318280
1263577	1310021	777245	2622798	129875094	139648668	115736982
47624	1251892	711774	2402149	129929228	139555942	115582109
213456	1282822	753668	2583958	131352225	140943272	116873090
241594	2237318	783813	3354161	132638328	142238840	118182686
47091	1256280	739676	2371092	131354918	140317676	116997801
213785	1287377	971298	2399864	133681598	142657486	119272040
242529	1328960	1001302	3178812	135958217	145079297	121754834
47625	1258558	698828	2401081	127994061	136911957	113350841
214409	1289145	741214	2598184	130259053	139832074	115665157
243646	2248716	771460	2626484	132557568	142084884	117914921
46994	1251945	705184	2398240	132300564	141340549	117929650
212342	1295788	746848	2595869	133595938	142634777	119221621
261558	2193920	777156	2623730	134884210	143924678	120506816
47212	1253158	73021	1610742	130798994	139774881	116098245
213549	1284160	99609	1744616	132089193	141066676	117383820
242441	2239674	123321	1776789	133367174	143444553	119806245

Таблиця 1.2 Результати часових замірів сліпону

Задача 1		**Задача 2**		**Задача 3**		
thread1	thread2	thread1	thread2	thread1	thread2	thread3
43564	725343	1316448	2976162	102078262	133712751	112901213
118123	769100	1338845	3046215	128891111	138009565	114353085
143660	1200000	1400000	3181333	130261010	135304936	113641163
32816	410276	1304494	2983214	123532401	131979408	111838656
60834	629711	1326657	3105448	126829346	134276972	114126333
90234	650112	1367262	3124678	129116185	139567492	115436474
31224	405218	1308370	2774049	123601001	134109194	110908506
64333	635432	1330576	3132768	127896732	137396498	113204681
90594	678000	1371333	3171152	129186172	139684248	115494587
43140	1259437	705546	2335544	130111125	137794753	114427274
210994	1270461	718757	2515304	131512121	140064645	115128480
244321	2267684	738890	2614334	132800000	141370294	116940230
43046	1231136	704033	2235170	127021000	136857211	112884123
193156	1246161	716017	2445092	128455440	138232345	113318250
1113577	1300011	757245	25122798	129873294	139640033	114736782
45624	1251333	701774	2392149	129924456	137554411	115580002
213426	1272822	733668	2583333	130351122	139943272	116551103
231689	2237211	753813	3354045	131630001	140238840	117181177
45091	1236280	739423	2331092	130354918	140315598	114997801
208785	1267343	901298	2396578	132681598	141657486	117272040
212231	1308768	991302	3018812	135954418	143079123	121734567
43625	1238533	678823	2391081	123994034	133911954	111350811
204409	1264356	711210	2518144	130250000	137832076	113665133
243234	2018717	731465	2596487	131557533	140084844	115914903
44994	1221945	705123	24598240	131300564	140340540	115929650
201346	1275788	716848	2545833	132595921	141634733	119221378
231554	2003920	737156	2603711	134880000	142924643	120506222
49212	1233158	71021	1600742	130792121	137774833	114098245
211549	1264240	93609	1714617	131089144	140066645	117383345
242422	2119674	123001	1776733	132367122	141444500	118806233

Таблиця 1.3 Результати часових замірів іменованого семафору

Задача 1		**Задача 2**		**Задача 3**		
thread1	thread2	thread1	thread2	thread1	thread2	thread3
35126	397308	1612694	3321769	2482152	4372596	8424286
462739	911047	7952690	9729318	10560942	16584384	16567858
34300	398401	1615338	3322437	2452602	4375234	8477848
465354	918640	7953460	9738997	10580548	16625484	16625484
33896	255886	1626348	2979876	2465784	4293396	8187692
318637	616536	7602878	9362326	10315484	16195208	14954034
33481	254582	1644689	2833734	2253274	2754930	4994828
317098	615853	6636198	7865826	7548118	12074096	13119216
33906	255480	1644476	2836286	2460110	4284618	8266356
317992	617184	6913534	8136114	10387420	16268676	15084038
34244	254516	1644434	2833322	2461678	4283074	7978086
318855	619492	6907009	8139378	10380550	15991948	14747734
34191	255934	1643860	2857332	2454516	4292482	8289468
318254	616142	6942889	8179268	10394326	16297708	15038470
33486	253027	1646568	2831206	2464030	4301732	8286676
316136	613970	6919016	8159576	10411712	16304844	15103462
33722	254746	1646282	2832148	2468588	4294392	8275336
316920	615034	6849321	12170380	10395380	16272944	15056050
33924	255732	1642962	2840038	2498536	4455894	8124752
318026	617348	6912560	8172801	10624958	16279026	14938890

Таблиця 1.4 Результати часових замірів неіменованого семафору

Задача 1		**Задача 2**		**Задача 3**		
thread1	thread2	thread1	thread2	thread1	thread2	thread3
22943	76906	1441386	3076129	2156256	4906692	3229862
398462	537371	3216669	6751721	4608990	11464290	11855126
21910	73760	1440466	3074249	2158240	4939756	3229198
381872	516835	3213664	6746588	4613246	11478546	11828076
22226	84810	1440462	3074553	2157260	4969864	4639694
254209	409242	3215810	6759572	4299850	13675726	12013792
21863	84806	1441412	3037302	2157338	4963424	3940608
255322	408835	3199698	6647102	4318652	12190148	10428172
22132	84971	1440385	3035390	2193242	4282860	2127182
254409	408303	3174896	6924180	4295150	8484406	8163502
22184	84769	1440940	3036034	2158804	4875424	3094948
253582	406901	3198465	6647277	4553000	10623042	10134112
21864	70819	1440897	3037176	2161554	4902138	3089804
267043	396203	3199741	6646173	4576342	10719838	10203304
21664	82899	1442066	3036701	2174832	4883632	3095196
253426	405334	3197884	6647365	4572292	10696032	10203428
22525	84405	1440829	3036113	2164286	4905434	3113582
254242	407836	3197921	6646208	4570062	10735016	10160520
21896	83800	1442934	3044585	2162396	4879656	3130026
254499	405606	3211394	6669852	4558354	10732826	10152848

Таблиця 1.5 Результати часових замірів мютексу

Задача 1		**Задача 2**		**Задача 3**		
thread1	thread2	thread1	thread2	thread1	thread2	thread3
21454	74023	1445457	3169428	2160506	4878904	2914512
378750	512618	2840237	6235258	4618416	12312164	10354764
21456	74020	1438424	3082561	2159640	4901552	2924318
394114	527822	3033448	6333486	4615634	12233044	10347768
21474	85718	1439700	3070145	2160676	4912616	2940932
222246	375536	3021561	6324089	4611086	9210560	7234592
21290	84128	1440181	3072513	2161320	4867564	2976306
219683	371116	3023761	6311904	4541076	10383862	9427480
21528	85070	1440074	3160904	2159562	4870990	2976870
219851	372504	2834329	6362186	4541426	10399622	9486900
21241	84514	1438938	3161160	2160308	4867584	3012024
222151	374351	2833570	6361722	4538770	10345694	8936630
21645	84822	1438704	3194884	2162740	4851034	3250454
218954	371166	3023077	6876098	4550976	11535062	9435196
21774	84604	1438389	3191918	2202248	5184432	2969868
218188	370286	3019353	6871353	4969992	10736096	9521738
22357	85103	1439074	3194814	2157526	4869778	2974332
218575	371122	3023429	6871138	4537446	10389112	9486494
22045	85956	1437898	3196082	2158814	4867794	2968174
229257	383052	3021594	6878294	4540792	10380148	9456914

4.3 Аналіз даних і результати

За результатами роботи програми було побудовано таблиці результатів для кожної з груп. Нижче на таблицях 1.6-1.8 наведені результати роботи першої групи.

Таблиця 1.6 Порівняння абсолютних результатів роботи примітивів першої групи

Задача 1					
Умовна змінна thread1	**Сліпон thread1**	**Різниця**	**Умовна зміна thread2**	**Сліпон thread2**	**Різниця**
47764	43564	9%	735844	725343	1.43%
120129	118123	2%	769254	769100	0.02%
148661	143660	3%	1267512	1200000	5.33%
33816	32816	3%	430276	410276	4.65%
63834	60834	5%	630741	629711	0.16%
91730	90234	2%	660025	650112	1.50%
34224	31224	9%	435218	405218	6.89%
64030	64333	0.5%	636716	635432	0.20%
92593	90594	2%	678269	678000	0.04%
47140	43140	8%	1259246	1259437	0.02%
215994	210994	2%	1290461	1270461	1.55%
244642	244321	0.1%	2237684	2267684	1.34%
47046	43046	9%	1236236	1231136	0.41%
198754	193156	3%	1266161	1246161	1.58%
1263577	1113577	12%	1310021	1300011	0.76%
47624	45624	4%	1251892	1251333	0.04%
213456	213426	0.01%	1282822	1272822	0.78%
241594	231689	4%	2237318	2237211	0.005%
47091	45091	4%	1256280	1236280	1.59%
213785	208785	2%	1287377	1267343	1.56%
242529	212231	12%	1328960	1308768	1.52%
47625	43625	8%	1258558	1238533	1.59%
214409	204409	5%	1289145	1264356	1.92%
243646	243234	0.2%	2248716	2018717	10.23%
46994	44994	4%	1251945	1221945	2.40%
212342	201346	5%	1295788	1275788	1.54%
261558	231554	11%	2193920	2003920	8.66%
47212	49212	4%	1253158	1233158	1.60%
213549	211549	1%	1284160	1264240	1.55%
242441	242422	0.01%	2239674	2119674	5.36%

Таблиця 1.7 Порівняння абсолютних результатів роботи примітивів першої групи.

Задача 2					
Умовна змінна	Сліпон		Умовна зміна	Сліпон	
thread1	**thread1**	**Різниця**	**thread2**	**thread2**	**Різниця**
1326449	1316448	0.75%	2986160	2976162	0.335%
1368890	1338845	2.19%	3156965	3046215	3.508%
1400157	1400000	0.01%	3189630	3181333	0.260%
1334494	1304494	2.25%	2984080	2983214	0.029%
1376657	1326657	3.63%	3155448	3105448	1.585%
1407262	1367262	2.84%	3184678	3124678	1.884%
1318370	1308370	0.76%	2974049	2774049	6.725%
1360576	1330576	2.20%	3172768	3132768	1.261%
1391208	1371333	1.43%	3201152	3171152	0.937%
706686	705546	0.16%	2395544	2335544	2.505%
748757	718757	4.01%	2595805	2515304	3.101%
778890	738890	5.14%	2624334	2614334	0.381%
704082	704033	0.01%	2395170	2235170	6.680%
746017	716017	4.02%	2595161	2445092	5.783%
777245	757245	2.57%	2622798	2512278	4.214%
711774	701774	1.40%	2402149	2392149	0.416%
753668	733668	2.65%	2583958	2583333	0.024%
783813	753813	3.83%	3354161	3354045	0.003%
739676	739423	0.03%	2371092	2331092	1.687%
971298	901298	7.21%	2399864	2396578	0.137%
1001302	991302	1.00%	3178812	3018812	5.033%
698828	678823	2.86%	2401081	2391081	0.416%
741214	711210	4.05%	2598184	2518144	3.081%

771460	731465	5.18%	2626484	2596487	1.142%
705184	705123	0.01%	2398240	2459820	2.568%
746848	716848	4.02%	2595869	2545833	1.928%
777156	737156	5.15%	2623730	2603711	0.763%
73021	71021	2.74%	1610742	1600742	0.621%
99609	93609	6.02%	1744616	1714617	1.720%
123321	123001	0.26%	1776789	1776733	0.003%

Таблиця 1.8 Порівняння абсолютних результатів роботи примітивів першої групи.

Задача 3								
Умовна змінна	Сліпон		Умовна зміна	Сліпон		Умовна зміна	Сліпон	
thread 1	**thread 1**	**Різниця**	**thread 2**	**thread 2**	**Різниця**	**thread 3**	**thread 3**	**Різниця**
127597828	102078262	20.0000%	136712777	133712751	2.1944%	112901868	112901213	0.0006%
128890090	128891111	0.0008%	138009565	138009565	0.0000%	114353085	114353085	0.0000%
130261122	130261010	0.0001%	139304936	135304936	2.8714%	115641333	113641163	1.7296%
128532401	123532401	3.8901%	137979408	131979408	4.3485%	113838753	111838656	1.7570%
129829346	126829346	2.3107%	139276972	134276972	3.5900%	115126044	114126333	0.8684%
131116185	129116185	1.5254%	140567492	139567492	0.7114%	116436474	115436474	0.8588%
129602517	123601001	4.6307%	139109194	134109194	3.5943%	114908506	110908506	3.4810%
130896	127896	2.2919	140396	137396	2.1368	116204	113204	2.5817

732	732	%	498	498	%	681	681	%
132186172	129186172	2.2695%	141684248	139684248	1.4116%	117495270	115494587	1.7028%
130228125	130111125	0.0898%	139794753	137794753	1.4307%	115527274	114427274	0.9522%
131514600	131512121	0.0019%	141064645	140064645	0.7089%	116828480	115128480	1.4551%
132804674	132800000	0.0035%	142370294	141370294	0.7024%	118140230	116940230	1.0157%
127021236	127021000	0.0002%	136857505	136857211	0.0002%	112885418	112884123	0.0011%
128455930	128455440	0.0004%	138233704	138232345	0.0010%	114318280	113318250	0.8748%
129875094	129873294	0.0014%	139648668	139640033	0.0062%	115736982	114736782	0.8642%
129929228	129924456	0.0037%	139555942	137554411	1.4342%	115582109	115580002	0.0018%
131352225	130351122	0.7622%	140943272	139943272	0.7095%	116873090	116551103	0.2755%
132638328	131630001	0.7602%	142238840	140238840	1.4061%	118182686	117181177	0.8474%
131354918	130354918	0.7613%	140317676	140315598	0.0015%	116997801	114997801	1.7094%
133681598	132681598	0.7480%	142657486	141657486	0.7010%	119272040	117272040	1.6768%
135958217	135954418	0.0028%	145079297	143079123	1.3787%	121754834	121734567	0.0166%
127994061	123994034	3.1252%	136911957	133911954	2.1912%	113350841	111350811	1.7645%
130259053	130250000	0.0069%	139832074	137832076	1.4303%	115665157	113665133	1.7291%
132557	131557	0.7544	142084	140084	1.4076	117914	115914	1.6962

568	533	%	884	844	%	921	903	%
132300564	131300564	0.7559 %	141340549	140340540	0.7075 %	117929650	115929650	1.6959 %
133595938	132595921	0.7485 %	142634777	141634733	0.7011 %	119221621	119221378	0.0002 %
134884210	134880000	0.0031 %	143924678	142924643	0.6948 %	120506816	120506222	0.0005 %
130798994	130792121	0.0053 %	139774881	137774833	1.4309 %	116098245	114098245	1.7227 %
132089193	131089144	0.7571 %	141066676	140066645	0.7089 %	117383820	117383345	0.0004 %
133367174	132367122	0.7498 %	143444553	141444500	1.3943 %	119806245	118806233	0.8347 %

Отримані дані демонструють, що в абсолютних величинах за результатами порівняння примітив синхронізації сліпон виконується в середньому на 0,8% відсотка більш ефективно ніж умовна змінна.

Для першої групи також пораховано відсотковий вклад роботи примітиву синхронізації до часу виконнання всієї програми. Отримані результати наведені в таблицях 1.9-1.15.

Таблиця 1.9 Порівняння виконання примітивів синхронізації першої групи першим потоком для першої задачі

Умовна змінна/сліпон	Приклад 1		
Час виконання примітива потоком 1	Потік 1	Потік 1	Різниця
Вибірка 1	1.32	1.40	6%
Вибірка 5	6.35	5.83	8%
Вибірка 10	1.23	1.25	2%
Результат всіх вибірок	1.02	0.99	3%

Таблиця 1.10 Порівняння виконання примітивів синхронізації першої групи другим потоком для першої задачі

Умовна змінна/сліпон	Приклад 1		
Час виконання примітива потоком 2	Потік 2	Потік 2	Різниця
Вибірка 1	11.26	11.70	4%
Вибірка 5	6.58	6.80	3%
Вибірка 10	11.34	10.95	3%
Результат всіх вибірок	7.38	7.39	0.2%

Таблиця 1.11 Порівняння виконання примітивів синхронізації першої групи першим потоком для другої задачі

Умовна змінна/сліпон	Приклад 2		
Час виконання примітива потоком 1	Потік 1	Потік 1	Різниця
Вибірка 1	4.68	4.74	1%
Вибірка 5	3.86	3.78	2%
Вибірка 10	0.99	1.01	2%
Результат всіх вибірок	4.02	3.96	1%

Таблиця 1.12 Порівняння виконання примітивів синхронізації першої групи другим потоком для другої задачі

Умовна змінна/сліпон	Приклад 2		
Час виконання примітива потоком 2	Потік 2	Потік 2	Різниця
Вибірка 1	10.65	10.77	1%
Вибірка 5	13.03	12.56	4%
Вибірка 10	14.27	14.62	2%
Результат всіх вибірок	12.14	18.93	56%

Таблиця 1.13 Порівняння виконання примітивів синхронізації першої групи першим потоком для третьої задачі

Умовна змінна/сліпон	Приклад 3		
Час виконання примітива потоком 1	Потік 1	Потік 1	Різниця
Вибірка 1	29.30	31.65	8%
Вибірка 5	29.17	29.98	3%
Вибірка 10	29.39	29.37	0.09%
Результат всіх вибірок	29.03	29.18	1%

Таблиця 1.14 Порівняння виконання примітивів синхронізації першої групи другим потоком для третьої задачі

Умовна змінна/сліпон	Приклад 3

Час виконання примітива потоком 2	Потік 2	Потік 2	Різниця
Вибірка 1	31.33	32.87	5%
Вибірка 5	31.37	32.24	3%
Вибірка 10	31.61	31.38	1%
Результат всіх вибірок	31.11	31.34	1%

Таблиця 1.15 Порівняння виконання примітивів синхронізації першої групи третім потоком для третьої задачі

Умовна змінна/сліпон	Приклад 3		
Час виконання примітива потоком 3	Потік 3	Потік 3	Різниця
Вибірка 1	26.01	27.61	6%
Вибірка 5	26.00	26.49	2%
Вибірка 10	26.40	26.36	0.17%
Результат всіх вибірок	25.82	26.08	1%

Результати порівняння відносного у (%) часу виконання примітиву синхронізації кожним з потоків показало доволі суперечливі результати. Незважаючи на те що в абсолютних числах перевага в швидкості виконання примітиву синхронізації сліпон була краща ніж в умовної змінної, але програми які використовували примітив синхронізації сліпон виконувалися довше ніж ті що використовували умовну зміну. Сліпон – це по суті в ОСРЧ QNX та сама умовна змінна, але з полегшеним доступом до її реалізації. Сліпон який є більш легким для розробника в використані примітивом на нашу думку виконується довше через те що система використовує більше часу для його створення та видалення ніж умовна змінна де розробник прописує всі складові реалізації примітиву самостійно.

Перейдемо до другої групи. Нижче на таблицях 1.16-1.22 наведені результати роботи першої групи.

Таблиця 1.16 Порівняння абсолютних результатів роботи примітивів другої групи

Задача 1					
1	2	3			

И. Семафор	Н. Семафор	Мютекс			
thread1	thread1	thread1	Дельта 1/2	Дельта 1/3	Дельта 2/3
35126	22943	21454	53%	64%	7%
462739	398462	378750	16%	22%	5%
34300	21910	21456	57%	60%	2%
465354	381872	394114	22%	18%	-3%
33896	22226	21474	53%	58%	4%
318637	254209	222246	25%	43%	14%
33481	21863	21290	53%	57%	3%
317098	255322	219683	24%	44%	16%
33906	22132	21528	53%	57%	3%
317992	254409	219851	25%	45%	16%
34244	22184	21241	54%	61%	4%
318855	253582	222151	26%	44%	14%
34191	21864	21645	56%	58%	1%
318254	267043	218954	19%	45%	22%
33486	21664	21774	55%	54%	-1%
316136	253426	218188	25%	45%	16%
33722	22525	22357	50%	51%	1%
316920	254242	218575	25%	45%	16%
33924	21896	22045	55%	54%	-1%
318026	254499	229257	25%	39%	11%

Таблиця 1.17 Порівняння абсолютних результатів роботи примітивів другої групи

Задача 1					
1	2	3			
И. Семафор	Н. Семафор	Мютекс			
thread2	thread2	thread2	Разн. 1/2	Разн. 1/3	Разн. 2/3

397308	76906	74023	417%	437%	3.9%
911047	537371	512618	70%	78%	4.8%
398401	73760	74020	440%	438%	-0.4%
918640	516835	527822	78%	74%	-2.1%
255886	84810	85718	202%	199%	-1.1%
616536	409242	375536	51%	64%	9.0%
254582	84806	84128	200%	203%	0.8%
615853	408835	371116	51%	66%	10.2%
255480	84971	85070	201%	200%	-0.1%
617184	408303	372504	51%	66%	9.6%
254516	84769	84514	200%	201%	0.3%
619492	406901	374351	52%	65%	8.7%
255934	70819	84822	261%	202%	-16.5%
616142	396203	371166	56%	66%	6.7%
253027	82899	84604	205%	199%	-2.0%
613970	405334	370286	51%	66%	9.5%
254746	84405	85103	202%	199%	-0.8%
615034	407836	371122	51%	66%	9.9%
255732	83800	85956	205%	198%	-2.5%
617348	405606	383052	52%	61%	5.9%

Таблиця 1.18 Порівняння абсолютних результатів роботи примітивів другої групи

Задача 2					
1	2	3			
И. Семафор	Н. Семафор	Мютекс			
thread1	thread1	thread1	Разн. 1/2	Разн. 1/3	Разн. 2/3
1612694	1441386	1445457	12%	12%	-0.3%
7952690	3216669	2840237	147%	180%	13.3%

1615338	1440466	1438424	12%	12%	0.1%
7953460	3213664	3033448	147%	162%	5.9%
1626348	1440462	1439700	13%	13%	0.1%
7602878	3215810	3021561	136%	152%	6.4%
1644689	1441412	1440181	14%	14%	0.1%
6636198	3199698	3023761	107%	119%	5.8%
1644476	1440385	1440074	14%	14%	0.02%
6913534	3174896	2834329	118%	144%	12.0%
1644434	1440940	1438938	14%	14%	0.1%
6907009	3198465	2833570	116%	144%	12.9%
1643860	1440897	1438704	14%	14%	0.2%
6942889	3199741	3023077	117%	130%	5.8%
1646568	1442066	1438389	14%	14%	0.3%
6919016	3197884	3019353	116%	129%	5.9%
1646282	1440829	1439074	14%	14%	0.1%
6849321	3197921	3023429	114%	127%	5.8%
1642962	1442934	1437898	14%	14%	0.4%
6912560	3211394	3021594	115%	129%	6.3%

Таблиця 1.19 Порівняння абсолютних результатів роботи примітивів другої групи

Задача 2					
1	2	3			
И. Семафор	Н. Семафор	Мютекс			
thread2	thread2	thread2	Разн. 1/2	Разн. 1/3	Разн. 2/3
3321769	3076129	3169428	8%	5%	-2.9%
9729318	6751721	6235258	44%	56%	8.3%
3322437	3074249	3082561	8%	8%	-0.3%
9738997	6746588	6333486	44%	54%	6.5%

2979876	3074553	3070145	-3%	-3%	0.1%
9362326	6759572	6324089	39%	48%	6.9%
2833734	3037302	3072513	-7%	-8%	-1.1%
7865826	6647102	6311904	18%	25%	5.3%
2836286	3035390	3160904	-7%	-10%	-4.0%
8136114	6924180	6362186	18%	28%	8.8%
2833322	3036034	3161160	-7%	-10%	-4.0%
8139378	6647277	6361722	22%	28%	4.5%
2857332	3037176	3194884	-6%	-11%	-4.9%
8179268	6646173	6876098	23%	19%	-3.3%
2831206	3036701	3191918	-7%	-11%	-4.9%
8159576	6647365	6871353	23%	19%	-3.3%
2832148	3036113	3194814	-7%	-11%	-5.0%
12170380	6646208	6871138	83%	77%	-3.3%
2840038	3044585	3196082	-7%	-11%	-4.7%
8172801	6669852	6878294	23%	19%	-3.0%

Таблиця 1.20 Порівняння абсолютних результатів роботи примітивів другої групи

Задача 3					
1	2	3			
И. Семафор	Н. Семафор	Мютекс			
thread1	thread1	thread1	Разн. 1/2	Разн. 1/3	Разн. 2/3
2482152	2156256	2160506	15%	15%	-0.2%
10560942	4608990	4618416	129%	129%	-0.2%
2452602	2158240	2159640	14%	14%	-0.1%
10580548	4613246	4615634	129%	129%	-0.1%
2465784	2157260	2160676	14%	14%	-0.2%
10315484	4299850	4611086	140%	124%	-6.7%

2253274	2157338	2161320	4%	4%	-0.2%
7548118	4318652	4541076	75%	66%	-4.9%
2460110	2193242	2159562	12%	14%	1.6%
10387420	4295150	4541426	142%	129%	-5.4%
2461678	2158804	2160308	14%	14%	-0.1%
10380550	4553000	4538770	128%	129%	0.3%
2454516	2161554	2162740	14%	13%	-0.1%
10394326	4576342	4550976	127%	128%	0.6%
2464030	2174832	2202248	13%	12%	-1.2%
10411712	4572292	4969992	128%	109%	-8.0%
2468588	2164286	2157526	14%	14%	0.3%
10395380	4570062	4537446	127%	129%	0.7%
2498536	2162396	2158814	16%	16%	0.2%
10624958	4558354	4540792	133%	134%	0.4%

Таблиця 1.21 Порівняння абсолютних результатів роботи примітивів другої групи

Задача 3					
1	2	3			
И. Семафор	Н. Семафор	Мютекс			
thread2	thread2	thread2	Разн. 1/2	Разн. 1/3	Разн. 2/3
4372596	4906692	4878904	-11%	-10%	0.6%
16584384	11464290	12312164	45%	35%	-6.9%
4375234	4939756	4901552	-11%	-11%	0.8%
16625484	11478546	12233044	45%	36%	-6.2%
4293396	4969864	4912616	-14%	-13%	1.2%
16195208	13675726	9210560	18%	76%	48.5%
2754930	4963424	4867564	-44%	-43%	2.0%
12074096	12190148	10383862	-1%	16%	17.4%

4284618	4282860	4870990	0%	-12%	-12.1%
16268676	8484406	10399622	92%	56%	-18.4%
4283074	4875424	4867584	-12%	-12%	0.2%
15991948	10623042	10345694	51%	55%	2.7%
4292482	4902138	4851034	-12%	-12%	1.1%
16297708	10719838	11535062	52%	41%	-7.1%
4301732	4883632	5184432	-12%	-17%	-5.8%
16304844	10696032	10736096	52%	52%	-0.4%
4294392	4905434	4869778	-12%	-12%	0.7%
16272944	10735016	10389112	52%	57%	3.3%
4455894	4879656	4867794	-9%	-8%	0.2%
16279026	10732826	10380148	52%	57%	3.4%

Таблиця 1.22 Порівняння абсолютних результатів роботи примітивів другої групи

Задача 3					
1	2	3			
И. Семафор	Н. Семафор	Мютекс			
thread3	thread3	thread3	Разн. 1/2	Разн. 1/3	Разн. 2/3
8424286	3229862	2914512	161%	189%	10.8%
16567858	11855126	10354764	40%	60%	14.5%
8477848	3229198	2924318	163%	190%	10.4%
16625484	11828076	10347768	41%	61%	14.3%
8187692	4639694	2940932	76%	178%	57.8%
14954034	12013792	7234592	24%	107%	66.1%
4994828	3940608	2976306	27%	68%	32.4%
13119216	10428172	9427480	26%	39%	10.6%
8266356	2127182	2976870	289%	178%	-28.5%
15084038	8163502	9486900	85%	59%	-13.9%

7978086	3094948	3012024	158%	165%	2.8%
14747734	10134112	8936630	46%	65%	13.4%
8289468	3089804	3250454	168%	155%	-4.9%
15038470	10203304	9435196	47%	59%	8.1%
8286676	3095196	2969868	168%	179%	4.2%
15103462	10203428	9521738	48%	59%	7.2%
8275336	3113582	2974332	166%	178%	4.7%
15056050	10160520	9486494	48%	59%	7.1%
8124752	3130026	2968174	160%	174%	5.5%
14938890	10152848	9456914	47%	58%	7.4%

В абсолютних числах за результатами моделювання трьох задач було встановлено що для першої задачі найефективнішим засобом синхронізації виявився м’ютекс, в другій та третій задачі було встановлено шо таких засобів синхронізації є два мютекс та неіменований семафор.

Для другої групи також пораховано відсотковий вклад роботи примітиву синхронізації до часу виконнання всієї програми. Отримані результати наведені в таблицях 1.23-1.29.

Таблиця 1.23 Порівняння виконання примітивів синхронізації другої групи першим потоком для першої задачі

Мютекс/им.семафор/неим.семафор	Пример 1					
Время выполнения примитива потоком 1	Поток 1	Поток 1	Поток 1	Δ н.сем/им.сем	Δ им. сем/ мютекс	Δ н.сем / мютекс
Выборка 1	0.66	0.64	0.68	7.3%	4%	3.3%
Выборка 5	0.62	0.62	0.63	1.2%	1%	2.5%
Выборка 10	0.640	0.62	0.639	3.3%	3%	0.1%
Результат всех выборок	0.356	0.347	0.361	4.2%	3%	1.5%

Таблиця 1.24 Порівняння виконання примітивів синхронізації другої групи другим потоком для першої задачі

Мютекс/им.семафор/неим.семафор	Пример 1

Время выполнения примитива потоком 2	Поток 2	Поток 2	Поток 2	Δ н.сем/им.сем	Δ мютекс/имсемафор	Δ н.сем/мютекс
Выборка 1	0.90	1.25	0.92	26.5%	29%	3%
Выборка 5	1.04	1.21	1.01	16.3%	14%	3%
Выборка 10	1.07	1.20	1.02	15.2%	11%	5%
Результат всех выборок	0.63	0.87	0.61	30.6%	28%	3%

Таблиця 1.25 Порівняння виконання примітивів синхронізації другої групи першим потоком для другої задачі

Мютекс/им.семафор/неим.семафор	Пример 2					
Время выполнения примитива потоком 1	Поток 1	Поток 1	Поток 1	Δ н.сем/им.сем	Δ мютекс/имсемафор	Δ н.сем/мютекс
Выборка 1	9.7	13.0	8.8	31.8%	25%	9%
Выборка 5	9.6	13.7	8.8	35.7%	30%	8%
Выборка 10	9.1	13.7	8.7	36.8%	34%	4%
Результат всех выборок	7.0	8.3	6.3	24.2%	15%	10%

Таблиця 1.26 Порівняння виконання примітивів синхронізації другої групи другим потоком для другої задачі

Мютекс/им.семафор/неим.семафор	Пример 2					
Время выполнения примитива потоком 2	Поток 2	Поток 2	Поток 2	Δ н.сем/им.сем	Δ мютекс/имсемафор	Δ н.сем/мютекс
Выборка 1	21.2	15.9	18.6	17.0%	34%	13%
Выборка 5	21.5	16.1	19.2	19.1%	33%	11%
Выборка 10	20.6	16.2	18.0	11.0%	27%	13%
Результат всех выборок	15.4	11.2	13.2	17.5%	37%	14%

Таблиця 1.27 Порівняння виконання примітивів синхронізації другої групи першим потоком для третьої задачі

Мютекс/им.семафор/неим.семафор	Пример 3					
Время выполнения	Пот	Пот	Пот	Δ	Δ мютекс/им	Δ

примитива потоком 1	ок 1	ок 1	ок 1	н.сем/им. сем	семафор	н.сем/мютекс
Выборка 1	12.6	20.5	10.6	48%	39%	16%
Выборка 5	15.3	20.6	11.9	42%	26%	22%
Выборка 10	12.4	21.0	11.8	44%	41%	5%
Результат всех выборок	10.7	11.9	9.0	24%	9%	16%

Таблиця 1.28 Порівняння виконання примітивів синхронізації другої групи другим потоком для третьої задачі

Мютекс/им.семафор/неим.семафор	Пример 3					
Время выполнения примитива потоком 2	Поток 2	Поток 2	Поток 2	Δ н.сем/им.сем	Δ мютекс/им семафор	Δ н.сем/мютекс
Выборка 1	33.6	32.3	26.4	18.1%	4%	21%
Выборка 5	28.3	32.2	23.2	27.9%	12%	18%
Выборка 10	28.3	32.1	27.7	13.8%	12%	2%
Результат всех выборок	22.1	20.1	20.2	0.5%	10%	8%

Таблиця 1.29 Порівняння виконання примітивів синхронізації другої групи третім потоком для третьої задачі

Мютекс/им.семафор/неим.семафор	Пример 3					
Время выполнения примитива потоком 3	Поток 3	Поток 3	Поток 3	Δ н.сем/им.сем	Δ мютекс/им семафор	Δ н.сем/мютекс
Выборка 1	28.2	32.2	27.3	15%	12%	3%
Выборка 5	25.8	29.9	22.4	25%	14%	13%
Выборка 10	25.8	29.5	26.2	11%	13%	2%
Результат всех выборок	17.4	23.1	17.5	24%	25%	1%

Результати порівняння відносного у (%) часу виконання примітиву синхронізації кожним з потоків показало доволі суперечливі результати. Незважаючи на те що в абсолютних числах перевага в швидкості виконання примітивів синхронізації мютекс та неіменований семафор була краща то за

результатами цього дослідження неможна зробити однозначні висновки про ефективність того чи іншого засобу синхронізації.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1Вимоги безпеки при виконанні робіт на робочому місці

Визначення поняття охорони праці дається в ст. 1 Закону України від 14 жовтня 1992 р. «Про охорону праці» [48]. Загальними словами охорона праці – це система суспільно-правових, економічних, лікувально-профілактичних та інших заходів спрямованих на максимальне збереження здоров’я, працездатності та загалом життя людини в процесі трудової діяльності.

Охорона праці в сфері інформаційних технологій регулюється за допомогою наступних наказів та нормативно-правових актів:

- НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» затверджені наказом Міністерства соціальної політики України від 14 лютого 2018 №207 [49].

- ДСанПіН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин затверджені постановою Головного державного санітарного лікаря України від 10.12.1998 № 7 [50].
- Примірна інструкція з охорони праці під час експлуатації електронно-обчислювальних машин, затверджена наказом Міністерства доходів і зборів України від 05.09.2013 № 443 [51].

В зазначених документах, зокрема в [49] містяться мінімальні вимоги безпеки при виконанні робіт з екранними пристроями за яким інженер розробник програмного забезпечення проводить більшу частину свого робочого часу. Серед основних вимог можна виділити наступні: щодня перед початком роботи необхідно візуально оглянути прилад та його комплектуючі на відсутність механічних пошкоджень, витирати прилад від пилу та інших забруднень, у разі виникнення аварійної ситуації відключити прилад від живлення (бажано щоб кімната була обладнана автоматичним вимикачем або мережевим фільтром, який знеструмить пошкоджений прилад без вимкнення освітлення). Забороняється виконувати зміни в конструкції та складі екранних пристроїв, виконувати ремонт і налагодження екранних пристроїв на робочому місці працівника під час безпосередньої роботи, працювати з екранними пристроями які мають несправності в роботі. Під час операторської роботи за екранними приладами мають дотримуватися оптимальні умови мікроклімату відповідно до ДСН 3.3.6.042-99 державних санітарних норм мікроклімату виробничих приміщень затверджених Головним державними санітарним лікарем України від 1 грудня 1999 року № 42 [52].

В ДСанПіН 3.3.2.007-98 [50] наведені вимоги до виробничих приміщень в яких розміщується електронно обчислювальна техніка. Основними вимогами є: площа на одне робоче місце має становити не менше ніж 6 м^2, а об’єм не менше ніж 20 м^3, приміщення мають бути обладнані аптечкою першої медичної

допомоги, робочі місця з екранними приладами слід розташовувати так щоб природне світло падало переважно зліва, при розміщенні робочих столів необхідно щоб відстань між двома екранними приладами становила не менше ніж 1,2 м. Висота робочої поверхні столу з екранним приладом має регулюватися в межах 680-800 мм, робочий стілець має бути зручним, підйомно-поворотним, регульованим за висотою, екран має розташовуватися на оптимальній відстані від очей користувача, що за нормою становить від 600 до 700 мм та під кутом 30° до нормальної лінії погляду працюючого. Клавіатура має бути розташована на поверхні столу на відстані від 100 до 300 мм від краю, який направлений до робітника, поверхня клавіатури має бути матовою з коефіцієнтом відбиття 0,4, розташування клавіатури має забезпечувати добру видимість екрана для зручності ручного керування.

5.2 Шкідливі виробничі фактори на робочому місці

Шкідливі виробничі фактори – фактори, тривалий вплив яких на працюючого у визначених умовах приведе до захворювання, зниження працездатності і негативного впливу на здоров'я нащадків. В залежності від рівня і тривалості впливу шкідливі фактори можуть класифікуватися і як небезпечні.

При роботі з програмою «QNX C++ SyncPrimComp» програміст багато часу проводить за комп’ютером.

Робота на комп’ютері пов’язана з такими шкідливими факторами:

- електромагнітні випромінювання;
- зорове навантаження;
- відблиски на екрані монітора;
- гіподинамія;
- порушення функцій опорного апарату;
- синдром зап’ястного каналу.

Для того щоб максимально знизити шкідливі фактори робоче місце програміста має відповідати нормам наведеним в документах [49-50].

Електромагнітне випромінювання має бути в межах гранично допустимих норм, які встановлені ДСН 239-96 державними санітарними нормами і правилами захисту населення від впливу електромагнітних випромінювань затверджених Міністерством охорони здоров'я від 1 серпня 1996 року №239 в редакції від 22 грудня 2017 року [55].

Проблеми із зором одна з найбільших проблем під час роботи за ЕОМ через те що зорова система людини погано пристосована до роботи з комп'ютерним зображенням. Екранне зображення відрізняється від природного тим, що воно:

- не цілісне, а складається з дискретних точок – пікселів;
- має значно менший контраст, який знижується за рахунок зовнішнього освітлення;
- зображення на екрані мерехтливе, що збільшує навантаження на зоровий апарат;
- зображення на екрані має значні перепади яскравості порівняно з оточуючим середовищем.

Щоб зменшити зорове напруження під час роботи за комп'ютером, рекомендується робити часті перерви. Наприклад, застосовуючи правило «20-6-10», тобто після 20 хвилин роботи за ПК треба подивитися на об'єкт віддалений від очей приблизно на 6 метрів протягом 10 секунд. Ще один спосіб полягає у виборі хорошого монітору з великою роздільною здатністю та з високою чіткістю зображення. Якщо монітор з певних причин не можна замінити на більш якісний, то можна застосовувати спеціальні комп'ютерні окуляри, які знижують навантаження на зоровий апарат.

Відблиски на екрані монітора, що виникають при неправильному освітленні, приводить до погіршення зору. З метою зниження рівня впливу на програміста даного шкідливого фактору, варто дотримуватись вимог державних будівельних норм ДБН В.2.5-28:2018 Природне і штучне освітлення

затвердженого наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України від 3 жовтня 2018 року №264 [56] або застосовувати рідкокристалічні монітори, які в силу своєї конструкції і використовуваних матеріалів мають менший коефіцієнт відбиття світла, тому відблисків на них практично не буває.

Головною профілактикою гіподинамії під час роботи є комплекс простих фізичних вправ (кругові оберти голови, присідання, випади вперед, кругові оберти суглобів рук та ніг) або піша прогулянка під час обідньої перерви.

Порушення функцій опорного апарату (сколіоз, кіфоз, лордоз) є поширеним захворюванням серед розробників програмного забезпечення. Основною профілактикою є комплекс спеціальних вправ та правильне облаштування робочого місця, зокрема робочого стільця згідно вимог документа [50].

Синдром зап’ястного каналу патологічний стан, що характеризується болем, відчуттям оніміння і поколювання в пальцях руки й самої кисті та виникає в результаті здавлювання серединного нерва в зап'ястковому каналі. Профілактикою даного захворювання є короткі перерви в роботі для виконання спеціального комплексу вправ для розминки кисті та суглобів рук та передпліччя або використання спеціального бандажу під час роботи.

Норми, які регламентують параметри мікроклімату виробничих приміщень детально наведені ДСН 3.3.6.042-99 [52]. Мікроклімат (метеорологічні умови) виробничого середовища суттєво впливає на стан організму працівника, його працездатність протягом робочого дня, зміни. Показники температури, відносної вологості, швидкості руху повітря, теплового випромінювання нагрітих поверхонь характеризують клімат внутрішнього середовища виробничого приміщення. Нижче наведено основні показники мікроклімату робочого приміщення та оптимальні норми, в межах яких вони мають знаходитися.

- Температура повітря в холодну пору року в приміщенні де відбувається робота становить 20 ºС, а в теплу пору 24 ºС (при оптимальних нормах в холодну пору року – 16 – 24 ºС; в теплу пору року – 18 – 25 ºС);
- Відносна вологість перебуває в діапазоні 45 – 55% (при оптимальних нормах 40 – 60%);
- Швидкість руху повітря – в холодну пору року 0,2 м/с (при нормі 0,1 – 0,3 м/с) , а в теплу пору 0,4 (при нормі 0,2 – 0,4 м/с).

Рівень шуму робочому приміщені регламентується наказом Міністерства охорони здоров'я «Про затвердження Державних санітарних норм допустимих рівнів шуму в приміщеннях житлових та громадських будинків і на території житлової забудови» від 22 лютого 2019 року №463 [55]. Рівень звуку в приміщені становить 40 дБа (при допустимій нормі в 50 дБа), критерії шуму становлять 40, що відповідає допустимій нормі.

Норми, які регламентують рівень виробничої загальної та локальної вібрації наведено в ДСН 3.3.6.039-99 Державні санітарні норми виробничої загальної та локальної вібрації затверджені постановою Головного санітарного лікаря України від 1 грудня 1999 року №39 [56]. Цей рівень для локальної вібрації в приміщенні в якому ведеться розробка становить 115 для віброшвидкості та 73 для віброприскорення протягом восьмигодинного робочого дня, що відповідає нормам [56]. Рівень загальної виробничої вібрації в офісі теж знаходиться в межах норми.

Приміщення в яких встановлені персональні комп'ютери, повинні мати природне та штучне освітлення відповідно до ДБН В.2.5-28:2018 [54]. Мінімальна освітленість встановлюється в залежності від розряду виконуваних зорових робіт. Для робітників сфери розробки програмного забезпечення вона прирівняється до IV розряду і складає 300...500 лк.

Розрахуємо освітленість робочого місця з ПК на відповідність розряду зорової роботи. За даними вимірювань рівень природної освітленості поверхні, де розташований ПК, складає 200 лк за освітленості тієї же поверхні відкритим небосхилом в 20000 лк, тобто КПО = 1%, що не відповідає нормативному КПО.

Для штучного освітлення у приміщенні використовуються люмінесцентні лампи.

Розрахунок штучного освітлення проведемо для кімнати площею 25 м^2, ширина якої складає 5м, довжина – 5м, висота – 3,5м.

Скористаємося методом використання світлового потоку. Для визначення потрібної кількості світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F = \frac{E*K*S*Z}{\eta} \quad (1)$$

F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; E = 300 Лк;

S – площа освітлюваного приміщення (у нашому випадку S=20м^2);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1,1... 1,2, в нашому випадку Z =1,1);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку K = 1,5);

η – коефіцієнт використання світлового потоку, (виражається відношенням світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп, і обчислюється в долях одиниці; залежить від характеристик

світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{ст.}$) і стелі ($\rho_{стелі}$)), значення коефіцієнтів дорівнюють $\rho_{ст}$ = 40% і $\rho_{стелі}$=60%.

Обчислимо індекс за формулою:

$$I = \frac{S}{h(A+B)} \quad (2)$$

S – площа приміщення, S = 25 м2;

h – розрахункова висота підвісу, h = 3,4 м;

A – ширина приміщення, A = 4 м;

B – довжина приміщення, B = 5 м.

$$I = \frac{25}{3{,}4 * (5+5)} = 0{,}74$$

Знаючи індекс приміщення I, за таблицею наведеною в документі [54] знаходимо $\eta = 0{,}24$.

Підставимо всі значення у формулу для визначення світлового потоку F:

$$F = \frac{300*1{,}5*25*1{,}1}{0{,}24} = 51563 \text{ Лм}$$

Для освітлення використані люмінесцентні лампи типу ЛБ 40-1, світловий потік яких F = 4320 Лм. Розрахуємо необхідну кількість ламп у світильниках за формулою:

$$N = \frac{F}{Fл}, \text{ де}$$

N – кількість ламп, що визначається;

F - світловий потік, F = 51563 Лм;

Fл- світловий потік лампи, Fл = 4320 Лм

$$N = \frac{51563}{4320} = 12$$

В приміщенні використовуються світильники типу ОД. Кожен світильник комплектується двома лампами. Тобто необхідно використовувати 6 світильників із 12 працюючими лампами в них (з яких три знаходяться безпосередньо над робочим місцем). Схема розташування світильників зображена на рисунку 1.34.

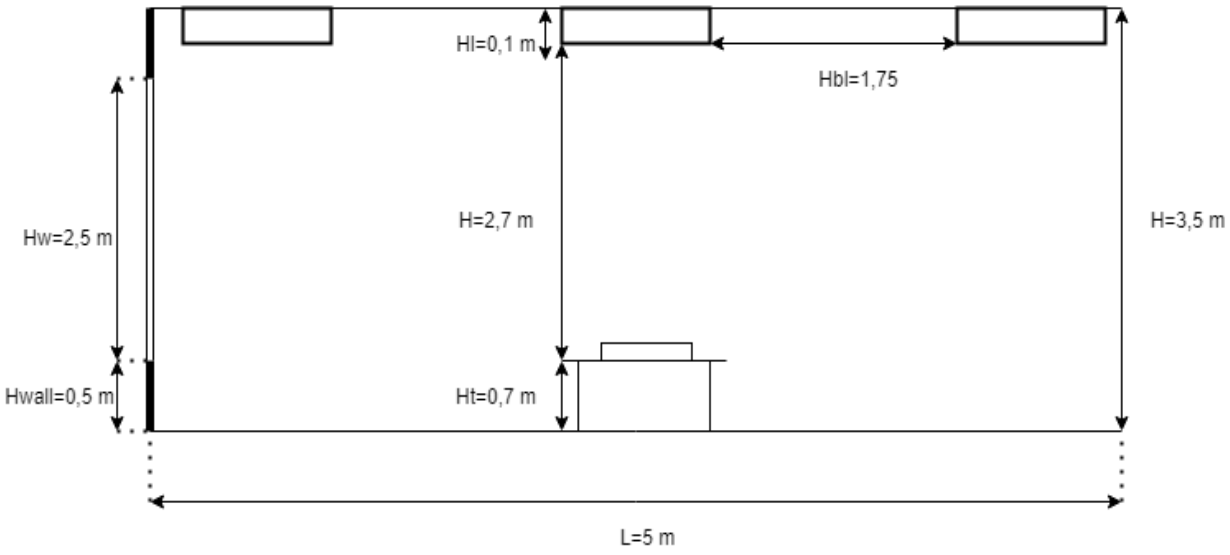


Рисунок 1.34 Схема розташування світильників

5.3 Дії працівників в аварійних ситуаціях

Дії працівників при аварійних ситуаціях, зокрема при електротравмі регламентуються Порядком надання домедичної допомоги постраждалим при ураженні електричним струмом та блискавкою затверджений Міністерством охорони здоров’я України від 16 червня 2014 №398 [57].

Заходи першої допомоги [58] залежать від стану потерпілого після визволення його від дії електричного струму. Для визначення стану необхідно вжити таких заходів:

- покласти потерпілого спиною на тверду поверхню;
- перевірити наявність у потерпілого дихання;
- перевірити наявність у потерпілого пульсу;
- з'ясувати стан зіниці, широка зіниця вказує на погіршення кровопостачання.

У всіх випадках ураження електричним струмом [59] виклик лікаря є обов'язковим незалежно від стану потерпілого.

Якщо потерпілий знаходиться при свідомості, його треба покласти у зручне положення і до прибуття лікаря забезпечити спокій, обов'язково спостерігаючи задиханням і пульсом. Не можна дозволяти потерпілому рухатись, продовжувати роботу [60-63].

Якщо потерпілий знаходиться у непритомному стані, його необхідно покласти, розстебнути одяг, забезпечити приплив свіжого повітря, дати понюхати нашатирний спирт, бризнути на нього водою і забезпечити спокій. Якщо потерпілий дихає погано, рідко і судомно, йому необхідно робити штучне дихання і непрямий масаж серця [64-70].

АНАЛІЗ РЕЗУЛЬТАТІВ ТА ВИСНОВКИ

Під час виконання роботи були оглянуті основні примітиви синхронізації, які використовуються в операційних системах реального часу, їх переваги і недоліки. Також, було проведено огляд існуючих комерційних продуктів, які використовуються для оцінки часу виконання програм. Було розроблено власний продукт, який отримує часові характеристики результатів роботи примітивів синхронізації, які статистично оброблюються та візуалізуються за допомогою Excel 2016.

Проектування системи відбувалося з використанням функціонального підходу, що дозволило зробити систему гнучкою, зрозумілою, надійною та такою що легко піддається модифікації. Реалізація системної функції ClockCycles() дозволила отримати максимально можливі часові характеристики роботи примітивів синхронізації. Для підвищення якості продукту, велику увагу було приділено тестуванню і налагодженню. Використання сучасних

інструментів тестування і відлагодження, а також використання різних підходів та методів тестування дозволили зробити систему більш надійною.

Для безпечної роботи з програмним продуктом, були досліджені шкідливі та небезпечні фактори, що можуть виникнути під час її функціонування. Були розроблені вимоги, прийоми та інструкції, що дозволять зменшити вплив цих факторів і зробити роботу з програмою ефективною і безпечною.

Проведене дослідження дозволило визначити ефективність впливу роботи примітиву синхронізації до загального часу виконання програми і дало відповідь щодо застосовності того чи іншого примітиву синхронізації в залежності від змодельованих умов.

Було визначено, що для першої групи в абсолютних величинах за результатами порівняння примітив синхронізації сліпон виконується в середньому на 0,8% відсотка більш ефективно ніж умовна змінна. Результати порівняння відносного у (%) часу виконання примітиву синхронізації кожним з потоків показало доволі суперечливі результати. Незважаючи на те що в абсолютних числах перевага в швидкості виконання примітиву синхронізації сліпон була краща ніж в умовної змінної, але програми які використовували примітив синхронізації сліпон виконувалися довше ніж ті що використовували умовну зміну. Сліпон – це по суті в ОСРЧ QNX та сама умовна змінна, але з полегшеним доступом до її реалізації. Сліпон який є більш легким для розробника в використані примітивом виконується довше через те що система використовує більше часу для його створення та видалення ніж умовна змінна де розробник прописує всі складові реалізації примітиву самостійно. Стосовно другої групи результати порівняння відносного у (%) часу виконання примітиву синхронізації кожним з потоків показало доволі суперечливі результати. Незважаючи на те що в абсолютних числах перевага в швидкості виконання примітивів синхронізації мютекс та неіменований семафор була краща то за результатами цього дослідження неможна зробити однозначні висновки про ефективність того чи іншого засобу синхронізації.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Ungurean I. «Timing Comparison of the Real-Time Operating Systems for Small Microcontrollers» [Ел. Ресурс]. Available: https://www.mdpi.com/2073-8994/12/4/592
   [Дата звернення: 15.11.2020].
2. Цырюлик О., Горшко Е., Зайцев В. QNX/UNIX: Анатомия параллелизма О. Цырюлик, Е. Горшко, В. Зайцев – Символ-Плюс, 2006. – 434 с.
3. Puschener P. Time analysis of security-critical real-time systems [Ел ресурс]. Available: https://ti.tuwien.ac.at/cps/teaching/courses/wcet
   [Дата звернення: 15.11.2020].
4. Wilhelm, R., Engblom, J., Ermedahl, A. «The worst-case execution time problem — overview of methods and survey of tools» – Transaction on Embedded Computing Systems, 2008.

5. Никольский О., Программирование приложений реального времени для исполнения в среде операционной системы реального времени QNX/NEUTRINO 2, О.Никольский – Брянский государственный технический университет: 2007. – 90 с.
6. Ermedahl, A. The worst-case execution time study&work – Transaction on Embedded Computing Systems, 2010.
7. Нечай В., Волошин Д., Нежуміра О. Дослідження граничних часових показників програмних середовищ ос реального часу, В.Нечай, Д.Волошин, О.Нежуміра [Ел ресурс]. Available: http://eadnurt.diit.edu.ua/handle/123456789/10684 [Дата звернення: 15.11.2020].
8. Махільов В. Результаты тестов производительности QNX Neutrino [Ел ресурс]. Available: https://www.cta.ru/cms/f/443753.pdf [Дата звернення: 15.11.2020].
9. QNX Neutrino [Ел ресурс]. Available: https://www.qnx.com/developers/docs/ [Дата звернення: 15.11.2020].
10. Butenhof D. Programing with POSIX Threads D. Butenhof Addison-Wesely: 2008. – 289 с.
11. Чекотовський Е. Статистичні методи на основі MS Excel 2013 Е. Чекотовський Знання: 2017. – 352 с.
12. «Медіана» [Ел ресурс]. Available: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B4%D1%96%D0%B0%D0%BD%D0%B0_(%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0). [Дата звернення: 18.11.2020].
13. «Максимум» [Ел ресурс]. Available: http://statistica.ru/glossary/general/maksimum/. [Дата звернення: 18.11.2020].

14. «Минимум» [Ел ресурс]. Available: http://statistica.ru/glossary/general/minimum/.
[Дата звернення: 18.11.2020].
15. «Размах вариации» [Ел ресурс]. Available: https://studbooks.net/53018/statistika/razmah_variatsii.
[Дата звернення: 18.11.2020].
16. «Среднее линейное отклонение» [Ел ресурс]. Available: https://univer-nn.ru/statistika/srednee-linejnoe-otklonenie/.
[Дата звернення: 20.11.2020].
17. «Дисперсия» [Ел ресурс]. Available: https://uk.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%B5%D1%80%D1%81%D1%96%D1%8F_%D0%B2%D0%B8%D0%BF%D0%B0%D0%B4%D0%BA%D0%BE%D0%B2%D0%BE%D1%97_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B8
[Дата звернення: 20.11.2020].
18. «Середньоквадратичне відхилення» [Ел ресурс]. Available: https://znaimo.com.ua/%D0%A1%D0%B5%D1%80%D0%B5%D0%B4%D0%BD%D1%8C%D0%BE%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%B8%D1%87%D0%BD%D0%B5%20%D0%B2%D1%96%D0%B4%D1%85%D0%B8%D0%BB%D0%B5%D0%BD%D0%BD%D1%8
[Дата звернення: 20.11.2020].
19. «Коефіцієнт варіації» [Ел ресурс]. Available: https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B5%D1%84%D1%96%D1%86%D1%96%D1%94%D0%BD%D1%82_%D0%B2%D0%B0%D1%80%D1%96%D0%B0%D1%86%D1%96%D1%97
[Дата звернення: 20.11.2020].
20. «Коефіцієнт осциліції» [Ел ресурс]. Available: https://kegt-rshu.in.ua/images/dustan/CMONS3.pdf
[Дата звернення: 20.11.2020].

21. «RapiTime» [Ел ресурс]. Available: https://www.rapitasystems.com/products/rapitime [Дата звернення: 22.11.2020].
22. «aiT» [Ел ресурс]. Available: https://www.absint.com/ait/index.htm [Дата звернення: 22.11.2020].
23. «Діаграма прецендентів» [Ел ресурс]. Available: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BF%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82%D1%96%D0%B2 [Дата звернення: 22.11.2020].
24. «UML Tutorial» [Ел ресурс]. Available: https://www.tutorialspoint.com/uml/index.htm [Дата звернення: 25.11.2020].
25. «Діаграма активностей» [Ел ресурс]. Available: https://flexberry.github.io/ru/fd_activity-diagram.html [Дата звернення: 25.11.2020].
26. «Діаграма станів» [Ел ресурс]. Available: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D1%81%D1%82%D0%B0%D0%BD%D1%96%D0%B2_(UML) [Дата звернення: 25.11.2020].
27. «Діаграма станів аспекти побудови» [Ел ресурс]. Available: http://it-gost.ru/articles/view_articles/97 [Дата звернення: 25.11.2020].
28. «Діаграма послідовності» [Ел ресурс]. Available: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BF%D0%BE%D1%81%D0%BB%D1%

96%D0%B4%D0%BE%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%96
[Дата звернення: 29.11.2020].

29. «Діаграма компонентів» [Ел ресурс]. Available: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D1%96%D0%B2
[Дата звернення: 29.11.2020].
30. «C язык программирования» [Ел ресурс]. Available: https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)
[Дата звернення: 29.11.2020].
31. «Unix» [Ел ресурс]. Available: https://www.opennet.ru/docs/RUS/unix_basic/
[Дата звернення: 30.11.2020].
32. «QNX Momentics» [Ел ресурс]. Available: https://www.swd.ru/index.php3?pid=396
[Дата звернення: 3.12.2020].
33. «Eclipse» [Ел ресурс]. Available: http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.ide.userguide%2Ftopic%2Fwhatsnew_whatsnewEclipse_.html
[Дата звернення: 3.12.2020].
34. «Функциональное программирование» [Ел ресурс]. Available: https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86
[Дата звернення: 3.12.2020].
35. «R programming language» [Ел ресурс]. Available: https://www.r-project.org/about.html
[Дата звернення: 3.12.2020].

36. «XML» [Ел ресурс]. Available: https://en.wikipedia.org/wiki/XML
[Дата звернення: 5.12.2020].
37. «Kotlin» [Ел ресурс]. Available: https://kotlinlang.org/
[Дата звернення: 7.12.2020].
38. «Perl» [Ел ресурс]. Available: https://www.perl.org/
[Дата звернення: 7.12.2020].
39. «Php» [Ел ресурс]. Available: https://htmlacademy.ru/tutorial/php
[Дата звернення: 10.12.2020].
40. «Python» [Ел ресурс]. Available: https://www.python.org/
[Дата звернення: 10.12.2020].
41. «Raku» [Ел ресурс]. Available: https://www.raku.org/
[Дата звернення: 10.12.2020].
42. «Scala» [Ел ресурс]. Available: https://www.scala-lang.org/
[Дата звернення: 12.12.2020].
43. «Рекурсія» [Ел ресурс]. Available: https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F
[Дата звернення: 12.12.2020].
44. «Ada» [Ел ресурс]. Available: https://www.adacore.com/about-ada
[Дата звернення: 12.12.2020].
45. «Тестування білим ящиком» [Ел. ресурс]. Available: https://ru.wikipedia.org/wiki/Тестирование_белого_ящика.
[Дата звернення: 1.12.2019].
46. «Тестування по стратегії чорного ящика» [Ел. ресурс]. Available: https://ru.wikipedia.org/wiki/Тестирование_по_стратегии_чёрного_ящика.
[Дата звернення: 1.12.2020].
47. «Стратегії тестування» [Ел. ресурс]. Available: http://dit.isuct.ru/Publish_RUP/core.base_rup/guidances/concepts/test_strategy_9981F03E.html.
[Дата звернення: 30.11.2020].

48. Закон України «Про охорону праці» від 14 жовтня 1992 р. Редакція від 16.10.2020

49. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» затверджені наказом Міністерства соціальної політики України від 14 лютого 2018 №207

50. ДСанПіН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин затверджені постановою Головного державного санітарного лікаря України від 10.12.1998 № 7

51. Примірна інструкція з охорони праці під час експлуатації електронно-обчислювальних машин, затверджена наказом Міністерства доходів і зборів України від 05.09.2013 № 443

52. ДСН 3.3.6.042-99 Державні санітарні норми мікроклімату виробничих приміщень затверджені постановою Головного державного санітарного лікаря України від 1 грудня 1999 року № 42

53. ДСН 239-96 Державні санітарні норми і правила захисту населення від впливу електромагнітних випромінювань затверджені Міністерством охорони здоров'я від 1 серпня 1996 року №239

54. ДБН В.2.5-28:2018 Державні будівельні норми України. Природне і штучне освітлення затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України від 3 жовтня 2018 року №264

55. Наказ Міністерства охорони здоров'я «Про затвердження Державних санітарних норм допустимих рівнів шуму в приміщеннях житлових та громадських будинків і на території житлової забудови» від 22 лютого 2019 року №463

56. ДСН 3.3.6.039-99 Державні санітарні норми виробничої загальної та локальної вібрації затверджені постановою Головного санітарного лікаря України від 1 грудня 1999 року №39

57. Порядок надання домедичної допомоги постраждалим при ураженні електричним струмом та блискавкою затверджений Міністерством охорони здоров'я України від 16 червня 2014 №398

58. Кодекс цивільного захисту України. Редакція від 16.10.2020

59. Кодекс законів про працю України від 10 грудня 1971 р. № 322-VIII. Редакція від 25.10.2020

60. Закон України від 23 вересня 1999 р. № 1105-XIV "Про загальнообов`язкове державне соціальне страхування". Редакція від 25.10.2020

61. Постанова Кабінету Міністрів України від 01 серпня 1992 р. № 442 "Про затвердження Порядку проведення атестації робочих місць за умовами праці". Редакція від 28.10.2016

62. Постанова Кабінету Міністрів України від 17.04.2019 №337 "Про затвердження Порядку розслідування та обліку нещасних випадків, професійних захворювань та аварій на виробництві"

63. НПАОП 0.00-4.12-05 Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці, затверджене наказом Держнаглядохоронпраці від 26.01.2005 №15. Зареєстрованого в Мін'юсті України 15.02.2005 за №231/10511. Редакція від 14.04.2017

64. НПАОП 0.00-7.14-17 Вимоги безпеки та захисту здоров'я під час використання виробничого обладнання працівниками, затверджене наказом Міністерства соціальної політики України від 28.12.2017 № 2072. Зареєстрованого в Мін'юсті України 23.01.2018 за №97/31549

65. ДСТУ 7299:2013 Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки, затверджено та введено в дію наказом міністерства економічного розвитку

і торгівлі України 14.10.2013 № 1231. Зареєстрованого в Мін'юсті України 14.02.2012 за №226/20539

66. «Перша медична допомога» [Ел. ресурс]. Available: https://moz.gov.ua/article/health/jak-nadati-pershu-dopomogu-zagalni-pravila. [Дата звернення: 1.12.2020].
67. «Перша медична долікарська допомога» [Ел. ресурс]. Available: https://www.bsmu.edu.ua/blog/6893-persha-medichna-dolikarska-dopomoga/. [Дата звернення: 1.12.2020].
68. «Інструкція з першої медичної допомоги» [Ел. ресурс]. Available: https://www.sop.com.ua/article/263-nstruktsya-z-nadannya-domedichno-dopomogi. [Дата звернення: 3.12.2020].
69. «Перша медична допомога при ураженні електричиним струмом» [Ел. ресурс]. Available: https://bozhedarivska-selrada.gov.ua/news/1576497483/. [Дата звернення: 3.12.2020].
70. «Перша медична допомога при ураженні електричиним струмом» [Ел. ресурс]. Available: https://lviv.dsp.gov.ua/operatyvna-informatsiia/novyny/pravyla-nadannia-pershoi-dopomohy-pry-u/. [Дата звернення: 3.12.2020].

71. ««VxWorks» [Ел. ресурс]. Available: https://embedded.prosoft.ru/products/brands/windriver/vxworks.html [Дата звернення: 5.12.2020].
72. «GNU Compiler Collection» [Ел. ресурс]. Available: https://uk.wikipedia.org/wiki/GNU_Compiler_Collection [Дата звернення: 5.12.2020].
73. «Умовна змінна» [Ел. ресурс]. Available: https://ru.wikipedia.org/wiki/%D0%A3%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F

[Дата звернення: 5.12.2020].

74. «Sleepon locks» [Ел. ресурс]. Available: http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.sys_arch%2Ftopic%2Fkernel_Sleepon_locks.html
[Дата звернення: 5.12.2020].

75. «Семафор» [Ел. ресурс]. Available: https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)
[Дата звернення: 5.12.2020].

76. «Мютекс» [Ел. ресурс]. Available: https://ru.wikipedia.org/wiki/%D0%9C%D1%8C%D1%8E%D1%82%D0%B5%D0%BA%D1%81
[Дата звернення: 5.12.2020].

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Б.Є. Боднар

СИСТЕМА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСОБІВ СИНХРОНІЗАЦІЇ ПОТОКІВ В ОС QNX

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01191-01-ЛЗ

Завідувач кафедри КІТ
проф. В.І. Шинкаренко
_____

Керівник розробки
доц. В.Я. Нечай
_____

Виконавець
студент групи ПЗ1921
Д.С.Сєнін
_____

Нормоконтролер
доц. О.С.Куропятник
_____

2020

ЗАТВЕРДЖЕНО
1116130.01191-01

СИСТЕМА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСОБІВ СИНХРОНІЗАЦІЇ ПОТОКІВ В ОС QNX

1116130.01191-01

Аркушів 22

2020

АНОТАЦІЯ

Документ 1116130.01191-01 «Система дослідження ефективності засобів синхронізації потоків в ОС QNX. Технічне завдання» входить до складу програмної документації до дипломного проекту.

У даному документі представлене призначення та область застосування програмного продукту, основні вимоги, стадії та строки виконання проекту, техніко-економічні показники, що пред’являються до програмного продукту.

ЗМІСТ

ВСТУП

На сьогоднішній день операційні системи реального часу взяли на себе виконання багатьох тривіальних задач зокрема в автоматизації процесів. При цьому всі задачі, що вирішуються можуть бути поділені на три категорії: задачі не критичні до часу їх вирішення, задачі м’якого реального часу (коли перевищення часу вирішення не бажано, але допустимо) і задачі жорсткого реального часу (коли перевищення часу вирішення може призвести до катастрофічних наслідків). Тому, оцінка граничних часових характеристик при розробці програм реального часу є обов’язковою процедурою.

Для отримання цієї граничної часової оцінки використовують три методи: динамічний, статичний і гібридний метод [3]. Найбільш точним, але і найменш універсальним вважається статичний метод. Але при додаванні певної межі похибки до оцінки, замість нього можна використовувати більш універсальний гібридний метод. Однак, для отримання цієї межі необхідно спочатку визначити вплив програмного середовища на час виконання програми.

Програмний продукт «QNX C++ SyncPrimComp» призначений для отримання часу виконання примітивів синхронізації операційної системи QNX для подальшої статистичної обробки цих даних в Excel. Дана обробка буде слугувати інструментом для отримання граничного часу реалізації примітивів синхронізації і подальшого аналізу цих оцінок для визначення впливу програмного середовища ОС на час виконання програм.

Підставою виникнення необхідності для розробки даного продукту стала відсутність прямих аналогів, а також неможливість модифікації існуючих подібних рішень.

Областю застосування програмного продукту може слугувати як сфера програмування. В сфері програмування, він може використовуватись для визначення граничного часу виконання примітивів синхронізації для операційних систем реального часу.

1 ПІДСТАВА ДО РОЗРОБКИ

Підставою для розробки є наказ ректора Дніпропетровського національного університету залізничного транспорту імені акад. В.Лазаряна Пшінька О.М. №779ст від 10.10.2019 р. «Про призначення наукових керівників та затвердження тем дипломних проектів» факультету «Технічна кібернетика» за спеціальністю 121 «Інженерія програмного забезпечення».

Тема проекту «Дослідження ефективності засобів синхронізації потоків в ОС QNX», керівник дипломного проекту доцент Нечай В.Я.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1 Функціональне призначення

Програмний продукт отримує часові характеристики виконання примітивів синхронізації операційної системи QNX і призначений для отримання граничної часової оцінки роботи примітиву синхронізації на мові C/C++ з використанням програми Excel.

2.2 Експлуатаційне призначення

Експлуатаційне призначення програмного продукту:

- інструмент для визначення часу виконання примітивів синхронізації в програмах, орієнтованих на ОС реального часу;
- інструмент статистичного аналізу часових характеристик примітивів синхронізації та їх вплив на найгірший час виконання програм Worst-case execution time (WCET), який реалізовано в програмі Excel.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Програмний продукт повинен відповідати наступним функціональним вимогам:

- введення і редагування вихідного коду;
- збереження в файл коду в редакторі;
- можливість компілювати і запускати вихідний код, при наявності відповідного компілятора;
- мінімальне налаштування інтерфейсу терміналу операційної системи;
- вивід на екран розрахованої часової оцінки виконання примітиву синхронізації;
- можливість перенести дані в текстовому файлі в Excel, де виконується статистична обробка отриманих даних.

Вхідні данні:

- код на мові C/C++, введений вручну в редакторі;
- текстовий файл, який можна перенести в середовище Excel для подальшої обробки;
- коментарі в коді з додатковою інформацію про вхідні дані програми, що аналізується, граничний час виконання окремих функцій.

Вихідні данні:

Результатом роботи програми є наступні вихідні дані:

- вихідний інструментований код C/C++;
- розрахований час виконання примітивів синхронізації, що реалізовані в програмі.

3.2 Вимоги до надійності

Усі вимоги, передбачені в даному розділі, мають на увазі надійність на рівні виконання програмних функцій і не враховують збоїв, що виникають з вини засобів обчислювальної техніки.

Вимоги до надійності програмного продукту наступні:

- наявність архівної копії тексту програми на зовнішньому носії;
- не більше ніж одна помилка на тисячу операторів;
- контроль коректності вхідної інформації, яка вводиться користувачем під час роботи з програмним продуктом.

3.3 Умови експлуатації

Даний програмний продукт може використовуватись в умовах, які відповідають вимогам [2].

Для нормального функціонування програмного продукту необхідно виконання наступних вимог:

- програмний комплекс повинен використовуватись в приміщеннях, призначених для роботи ЕОМ з наступними кліматичними умовами: температура – 21-25 °С, відносна вологість повітря 40-60%;
- кваліфікація персоналу повинна бути на рівні користувача ЕОМ, маючого досвід роботи з операційною системою Windows та ознайомленим з керівництвом користувача.

3.4 Вимоги до складу та параметрів технічних засобів

Мінімальна конфігурація комп’ютеру для забезпечення роботи програмного продукту:

- IBM-сумісний комп’ютер з тактовою частотою процесора 1,3 ГГц;
- ОЗП не менш ніж 512 Мб;

- вільний дисковий простір 200 Мб;
- наявність CD/DVD приводу, USB роз’єму для встановлення ПЗ;
- монітор з роздільною здатністю екрану 1024x768 та більше;
- стандартні клавіатура та маніпулятор «миша».

3.5 Вимоги до інформаційної та програмної сумісності

Для функціонування програмного продукту необхідні:

- ОС QNX 6 або більш пізня версія;
- QNX Momentics IDE.

3.6 Вимоги до маркування та упаковки

Програма може зберігатися на змінних носіях (CD/DVD-диски). Упаковка продукту повинна забезпечувати захист від механічних пошкоджень. Упаковка повинна мати маркування:

"QNX C++ SyncPrimComp", 1.0
Сєнін Дмитро Святославович
кафедра КІТ, ДНУЗТ 2020

3.7 Вимоги до транспортування та зберігання

Умови транспортування та зберігання повинні забезпечувати захист носія від фізичних пошкоджень і відповідати вимогам [2].

Строк зберігання програмного продукту необмежений та залежить напряму від умов зберігання.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмна документація повинна включати наступні документи: технічне завдання та робочий проект у складі:

- специфікація;
- текст програми;
- опис програми;
- керівництво користувача.

Вся документація до програми повинна задовольняти вимогам державного стандарту до оформлення програмних документів (ГОСТ 19.101-77).

5 ВИЗНАЧЕННЯ ВИТРАТ НА ПРОЕКТУВАННЯ ПРОГРАМИ

Основна мета розробки техніко-економічного обґрунтування (ТЕО) – це оцінка фінансових витрат на розробку проекту, прогнозування можливості реалізації та окупності проекту.

Початковим етапом розрахунку величини трудових витрат розробників є оцінка розміру програмного забезпечення. Основні відмінності методик, що застосовуються в оцінці трудовитрат, полягають у використовуваному типі критерію оцінки якості (кількісний або якісний) [1].

Згідно моделі COCOMO, розмір проекту S вимірюється в рядках коду LOC (KLOC), а трудовитрати в людино-місяцях.

$$E = a \cdot S^b \cdot EAF \quad (1)$$

де E – витрати праці на проект (в людино-місяцях);

S^b – розмір коду (в KLOC);

EAF – фактор уточнення витрат (effort adjustment factor).

Для простих систем, $a = 2{,}4$; $b = 1{,}05$ [1].

Розмір програмного коду програмного засобу для дослідження засобів синхронізації потоків в ОС QNX «QNX C++ SyncPrimComp» – 650 рядків:

$$E = 2{,}4 \cdot 0{,}65^{1{,}05} \cdot 1 = 1{,}53 \quad (2)$$

Отже, згідно моделі COCOMO, орієнтовні трудовитрати на проект складуть приблизно 1,53 людино-місяці.

Нижче наведені розрахунки вартості розробки програми «QNX C++ SyncPrimComp». Основними статтями витрат прийняті:

- основна заробітна плата;
- відрахування на соціальні потреби;
- накладні витрати;
- витрати на персональний комп’ютер і ліцензійні базові програмні засоби.

Основна заробітна плата (ОЗП) – це показник, який визначає розмір фінансової винагороди працівникам за виконану роботу. Він визначається

виходячи з кількості працівників, часу виконання роботи (годин) і заробітної плати в розрахунку на одну годину. Рекомендована кількість інженерів-програмістів для розробки даного проекту – 1 чол; тривалість розробки – 1,5 місяці. Середньомісячна заробітна плата працівників у сфері інформаційних та телекомунікаційних технологій за 2020р. наведена в табл. 5.1.

Таблиця 5.1 – Середньомісячна заробітна плата працівників у сфері інформації та телекомунікаційних технологій за період з початку 2020 р.*

Вид діяльності	Нараховано в середньому працівнику, грн								
	січень	лютий	березень	квітень	травень	червень	липень	серпень	вересень
Усього	15787	16447	17990	15278	15191	16393	17053	16795	17253

* – довідкова інформація Державної служби статистики України [2].

З цих даних можна розрахувати середню заробітну плату інженера-програміста:

$$\frac{15787 + 16447 + 17990 + 15278 + 15191 + 16393 + 17053 + 16795 + 17253}{9} =$$

= 16465 грн./міс.

Розрахунок заробітної платні проводиться по формі табл. 5.2.

Таблиця 5.2 – Фонд заробітної плати на період розробки

№ п/п	Посада виконавця	Оклад, грн/міс	Кількість		Сума зарплати грн
			чол	місяців	
1	інженер-програміст	16465	1	1,53	28812

Описаний в проекті програмний продукт буде розроблений одним програмістом в період з 5.07.20 до 23.08.20, що складає 35 робочих днів або 7 робочих тижнів. Витрати робочого часу прийняті за 40 годин у тиждень та відповідно, 160 годин на місяць. Погодинна ставка кваліфікованого інженера–

програміста складає $\frac{16465,22}{160} = 102,9$ грн/год. Таким чином, витрачено робочого часу:

$$t_{\text{розробки}} = N_{\text{чол}} \cdot N_{\text{тиж}} \cdot N_{\text{год}}, \quad (3)$$

де $N_{\text{чол}}$ – кількість виконавців, *чол*;

$N_{\text{тиж}}$ – тривалість розробки;

$N_{\text{год}}$ – витрати робочого часу, год;

$$t_{\text{розробки}} = 1 \cdot 7 \cdot 40 = 280\text{чол/год}. \quad (4)$$

ОЗП визначається за формулою:

$$\text{ОЗП} = t_{\text{розробки}} \cdot N \cdot K_{KB}, \quad (5)$$

Де $t_{\text{розробки}}$ – витрати праці у чол/год*;*

N – погодинна ставка;

K_{KB} – коефіцієнт кваліфікації програміста, приймається 0,8 [3].

ОЗП складає:

$$\text{ОЗП} = 280 \cdot 102,9 \cdot 0,8 = 23050\text{грн}. \quad (6)$$

Відрахування на соціальні потреби встановлюються у відсотках від суми заробітної плати [4]:

$$C_{\text{соц}} = \frac{\text{ОЗП} \cdot 22\%}{100\%}$$

$$C_{\text{соц}} = \frac{23050 \cdot 22\%}{100\%} = 5071\text{грн}. \quad (7)$$

Отримані результати за (6) та (7) підсумовуються. Вони складають 28121 грн. та визначають основні прямі витрати.

Накладні витрати враховують загальногосподарчі витрати по забезпеченню проведення роботи: витрати на опалення, електроенергію, водопостачання та водовідведення, амортизація будівель, зарплату адміністративного персоналу та інше. Вони визначаються в процентах (30 – 40%) від суми прямих витрат:

$$\text{С}_{\text{накл}} = \frac{(\text{ОЗП} + \text{С}_{\text{соц}}) \cdot 30\%}{100\%}; \quad (8)$$

$$С_{накл} = \frac{28121 \cdot 30\%}{100\%} = 8436 грн. \quad (9)$$

На протязі усього терміну використання нової техніки підприємство щорічно витрачає певні кошти, пов’язані з її експлуатацією.

Експлуатаційні витрати на персональний комп’ютер визначаються протягом терміну розробки програмного засобу в залежності від вартості комп’ютеру. В експлуатаційні витрати входять:

– вартість витратних матеріалів;
– витрати на ремонт;
– заробітна плата ремонтника;
– оренда приміщення;
– додаткові витрати – прибирання приміщення, охорона, оренда, комунальні послуги;
– амортизаційні витрати на персональний комп'ютер і програмне забезпечення;
– витрати на електроенергію ($С_{ел}$),

які визначаються за формулою:

$$С_{ел} = P \cdot B \cdot T_{розр}, \quad (10)$$

де Р – потужність комп’ютера та допоміжних споживачів електричної енергії, приймається 0,5 кВт/год [5];

B – вартість 1 кВт/година, складає 1,8 грн [6];

$T_{розр}$ – час роботи з ЕВМ, приймається рівним робочому часу.

Витрати на електроенергію визначаються так:

$$C_{ел} = 0{,}5 \cdot 1{,}84 \cdot 280 = 258 грн. \quad (11)$$

Витрати на витратні матеріали ($С_{вм}$) протягом всього терміну експлуатації приблизно 10% від вартості комп’ютеру. Вартість робочої станції приймається 33 999 грн., термін експлуатації – 5 років. Отже, можна визначити ці витрати за період створення програмного засобу [7]:

$$С_{вм} = В_{ком} \cdot \frac{N_Д}{N_{експ} \cdot 365} \cdot \frac{10\%}{100\%}, \quad (12)$$

де $B_{КОМ}$ – вартість персонального комп'ютеру;

$N_{Д}$ – кількість днів розробки програмного продукту;

$N_{експ}$ – термін експлуатації персонального комп'ютеру.

Витрати на витратні матеріали визначаються так:

$$C_{вм} = 33999 \cdot \frac{35}{5 \cdot 365} \cdot \frac{10}{100} = 65 \text{ грн.} \quad (13)$$

Заробітна плата ремонтника ($C_{рем}$) визначена наступним чином: на ремонт 50 комп'ютерів потрібен один інженер-системотехнік. Його середньомісячна заробітна плата приймається 16465 грн. Тоді в перерахунку на один комп'ютер його заробітна плата за період розробки програмного продукту складає:

$$C_{рем} = \frac{C_{рем}'}{N_{КОМ}} \cdot T_{міс}, \quad (14)$$

де $C_{рем}'$ – середньомісячна заробітна плата;

$N_{КОМ}$ – кількість комп'ютерів на одного ремонтника.

$T_{міс}$ – час розробки програмного продукта, міс.

Заробітна плата ремонтника ($C_{рем}$) буде складати:

$$C_{рем} = \frac{16465,22}{50} \cdot 1,75 = 576 \text{ грн.}$$

За статистикою витрати на комплектуючи вироби ($C_{КОМ}$) для ремонту персонального комп'ютера складає 10% від його вартості за термін його експлуатації, тобто рівні витратам на витратні матеріали:

$$C_{КОМ} = C_{ВМ} = 65 \text{ грн.} \quad (16)$$

Амортизаційні відрахування на персональний комп'ютер (АПК) визначені з положення, що амортизаційний період в даний час дорівнює терміну морального старіння обчислювальної техніки і складає 3 роки. Отже, за 3 роки амортизаційні відрахування на персональний комп'ютер дорівнюють вартості комп'ютера. За період проектування амортизаційні відрахування складуть:

$$АКП = В_{КОМ} \cdot \frac{N_{Д}}{N_{експ} \cdot 365}; \quad (17)$$

$$АКП = 33999 \cdot \frac{1{,}75}{3 \cdot 12} = 1653$$

Амортизаційні відрахування на програмне забезпечення (АПЗ) залежать від його циклу заміни. Якщо прийняти термін морального старіння для ОС QNX 5 років та IDE QNX Momentics, що входить до складу операційної системи за 5 років, а також Microsoft Excel за 3 роки то амортизаційні відрахування на програмне забезпечення дорівнюють його вартості.

Для функціонування персонального комп'ютера використовувалася операційна система QNX 4.25, для написання програмного забезпечення - програмне середовище IDE QNX Momentics, для обробки статистичних даних Excel 2016.

$$АКП_q = 33742 \cdot \frac{1{,}75}{5 \cdot 12} = 984$$

$$АКП_e = 4801 \cdot \frac{1{,}75}{3 \cdot 12} = 233$$

Розрахунок амортизаційних відрахувань на програмне забезпечення зведений в табл. 5.2.

Таблиця 5.2 – Використовуване програмне забезпечення

Найменування програмного забезпечення	Вартість програмного забезпечення, *грн*	Джерело придбання	Амортизаційні відрахування, *грн*
ОС QNX 4.25 (з середовищем розробки програм)	33742	http://www.rts.ua/rus/catshop/473/0/4806/qnx/	984
Excel 2016	4801	https://www.softmart.ua/microsoft-excel-2016.html	233
Всього:	38543	-	1217

Додаткові витрати ($C_{\text{дод}}$): прибирання приміщень, охорона, комунальні послуги важко оцінити точно і прийняти рівними 50% заробітної плати інженера-програміст, тобто 8233 гривень на місяць.

Оренду приміщення розрахуємо так: норма площі на одне робоче місце становить не менше 6 м^2, орендувати такий офіс можна за 2475 гривень на місяць [8-9].

Сумарні експлуатаційні витрати на один персональний комп'ютер складають:

$$C_{\text{експ}} = C_{\text{ел}} + C_{\text{ВМ}} + C_{\text{рем}} + \text{АПК} + \text{АПО} + C_{\text{ор}} + C_{\text{дод}}; \quad (18)$$

$$C_{\text{експ}} = 258 + 65 + 576 + 65 + 1653 + 1217 + 4950 + 8233 = \\ = 17017\text{грн}. \quad (19)$$

Результати розрахунків зведено у табл. 5.3.

Таблиця 5.3 – Експлуатаційні витрати на ПК і ПЗ.

Найменування витрат	Витрати, грн
Витрати на електроенергію	258
Вартість витратних матеріалів	65
Витрати на ремонт	641
Амортизація персонального комп'ютера	1653
Амортизація програмного забезпечення	1217
Оренда приміщення	4950
Додаткові витрати	8233
Всього	17017

Таким чином, витрати на створення програмного продукту складають:

$$C_{\text{розробки}} = \text{ОЗП} + C_{\text{соц}} + C_{\text{накл}} + C_{\text{експ}}; \quad (20)$$

$$C_{\text{розробки}} = 23050 + 5071 + 8436 + 17017 = \\ = 53574 \text{ грн} \quad (21)$$

Розрахунок витрат зведено у табл. 5.4.

Таблиця 5.4 – Кошторис витрат на розробку програмного засобу

Найменування витрат	Витрати, грн
Основна заробітна плата	23050
Відрахування на соціальні потреби	5071
Накладні витрати	8436
Експлуатаційні витрати	17017
Всього	53574

За отриманими значеннями техніко-економічних показників проекту складено кошторис витрат на розробку сучасного програмного забезпечення для дослідження засобів синхронізації потоків в ОС QNX «QNX C++ SyncPrimComp». За результатами розрахунків, приблизна вартість розробки складає 53574 грн.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Стадії та етапи розробки програмного продукту представлені у табл. 6.1.

Таблиця 6.1 – Стадії та етапи розробки

Стадії розробки	Етапи розробки	Терміни виконання
1. Технічне завдання (ТЗ)	Функціональне та експлуатаційне призначення	24.09.20 – 25.09.20
	Вхідні та вихідні дані	25.09.20 – 26.09.20
	Вимоги до програмної документації	26.09.20 – 28.09.20
	Техніко-економічні показники	28.09.20 – 30.09.20
	Узгодження та затвердження ТЗ	01.10.20 – 02.10.20
2. Робочий проект	Розробка та програмування логіки програми	05.10.20 – 25.10.20
	Розробка і реалізація інтерфейсу користувача	25.10.20 – 26.11.20
	Налагодження програми	26.11.20 – 01.12.20
	Розробка, узгодження та затвердження програмної документації відповідно до вимог ГОСТ 19.101-77	01.12.20 – 04.12.20
	Проведення випробувань і корегування програми та документації за результатами випробувань	04.12.20 – 07.12.20
3. Впровадження	Підготовка і передача програми та програмної документації замовнику	08.12.20 – 10.12.20

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Контроль здійснюється за допомогою виконання набору тестів з метою знаходження помилок в програмному продукті та його специфікації. Контроль виконання роботи забезпечується головним керівником розробки доц., к.т.н Нечаєм В.Я.

Прийом програмного продукту здійснюється уповноваженою комісією.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. – 38 с.
2. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2.007-98
3. Haugli, F.B. «Using online worst-case execution time analysis and alternative tasks in real time systems» – NTNU-Trondheim, 2014. – 84 p.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Б.Є. Боднар

СИСТЕМА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСОБІВ СИНХРОНІЗАЦІЇ ПОТОКІВ В ОС QNX

Робочий проект
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01191-01-ЛЗ

Завідувач кафедри КІТ
проф. В.І. Шинкаренко
_____

Керівник розробки
доц. В.Я. Нечай
_____

Виконавець
студент групи ПЗ1921
Д.С. Сєнін
_____

Нормоконтролер
доц. О.С. Куроп’ятник
_____

2020

ЗАТВЕРДЖЕНО
1116130.01191-01 13 01

СИСТЕМА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСОБІВ СИНХРОНІЗАЦІЇ ПОТОКІВ В ОС QNX

Опис програми

1116130.01191-01 13 01

Аркушів 18

2020

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмна система «QNX C++ SyncPrimComp» отримує часові характеристики примітивів синхронізації в вигляді текстового файлу, який потім статистично обробляється для визначення оцінки граничного часу виконання примітиву синхронізації, орієнтованих на операційні системи реального часу. Програма надає програмісту можливість зрозуміти вклад кожного примітиву синхронізації в найгірший час виконання програм залежно від поставлених критеріїв.

Програма функціонує в середовищі QNX 6 та у більш пізніх версіях операційної системи QNX. Програма розроблена на мові С в середовищі QNX Momentics IDE та оброблена в Excel 2016.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Програмний продукт отримує часові характеристики виконання примітивів синхронізації операційної системи QNX і призначений для отримання граничної часової оцінки роботи заданих примітивів синхронізації на мові C/C++ з використанням програми Excel для статистичної обробки.

Областю застосування програмного продукту може слугувати як сфера програмування. В сфері програмування, він може використовуватись для визначення граничного часу виконання примітивів синхронізації для операційних систем реального часу.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1 Алгоритм програми

На рис. 3.1 та 3.2 зображено алгоритм роботи програми.

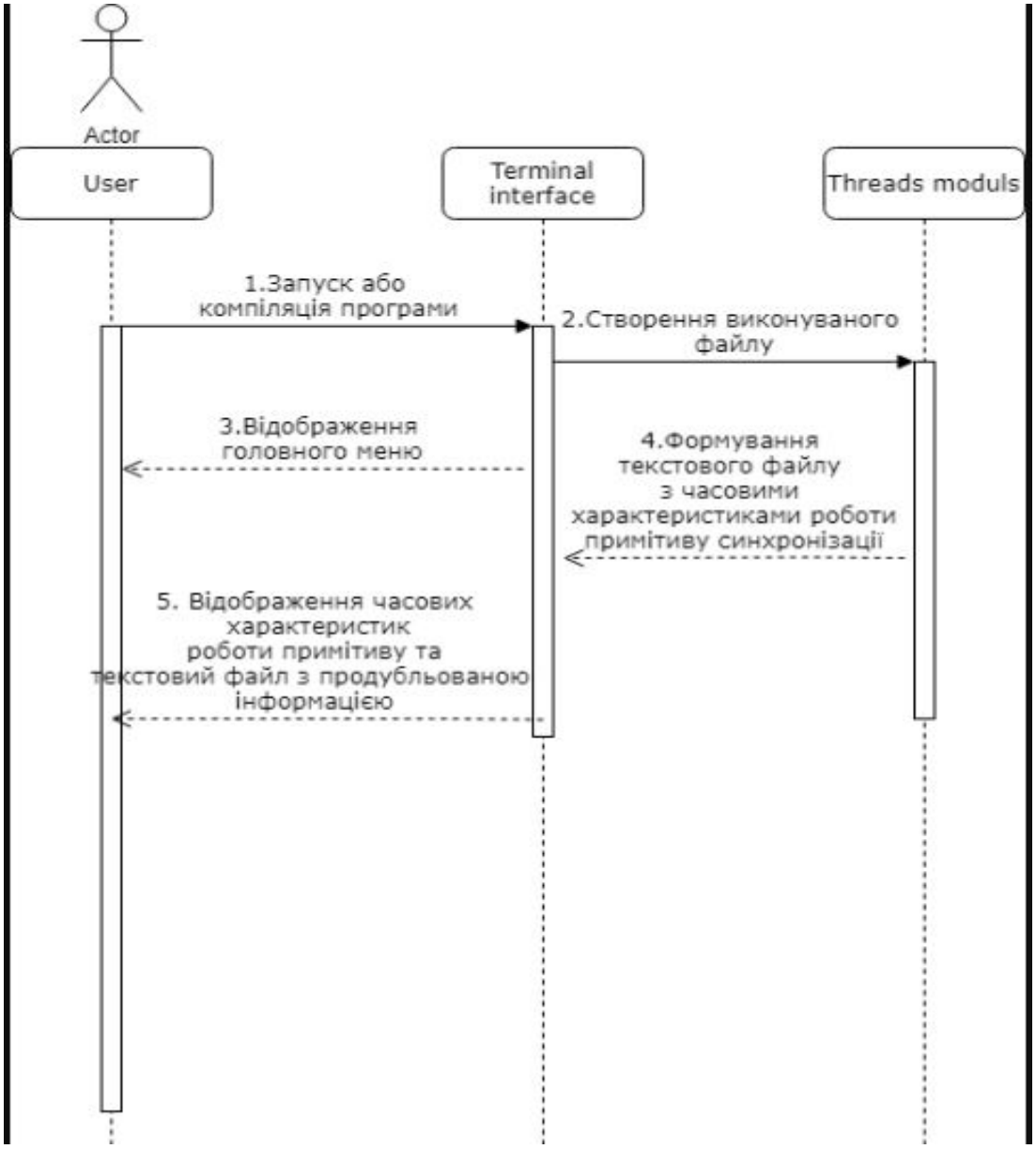


Рисунок 3.1 – Алгоритм роботи програми

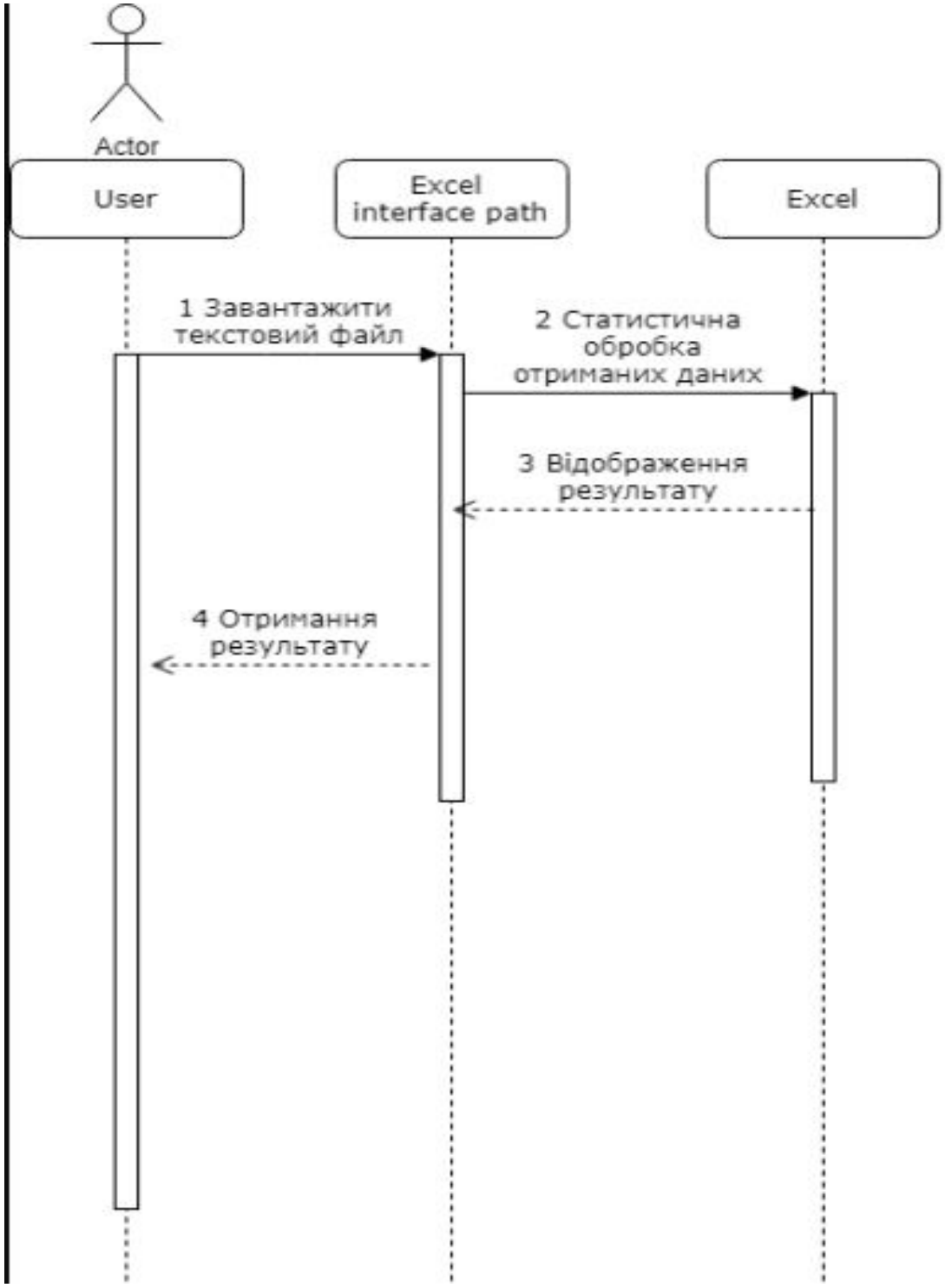


Рисунок 3.2 – Алгоритм роботи програми

На рис. 3.3 та 3.4 зображено алгоритм послідовності дій програми.

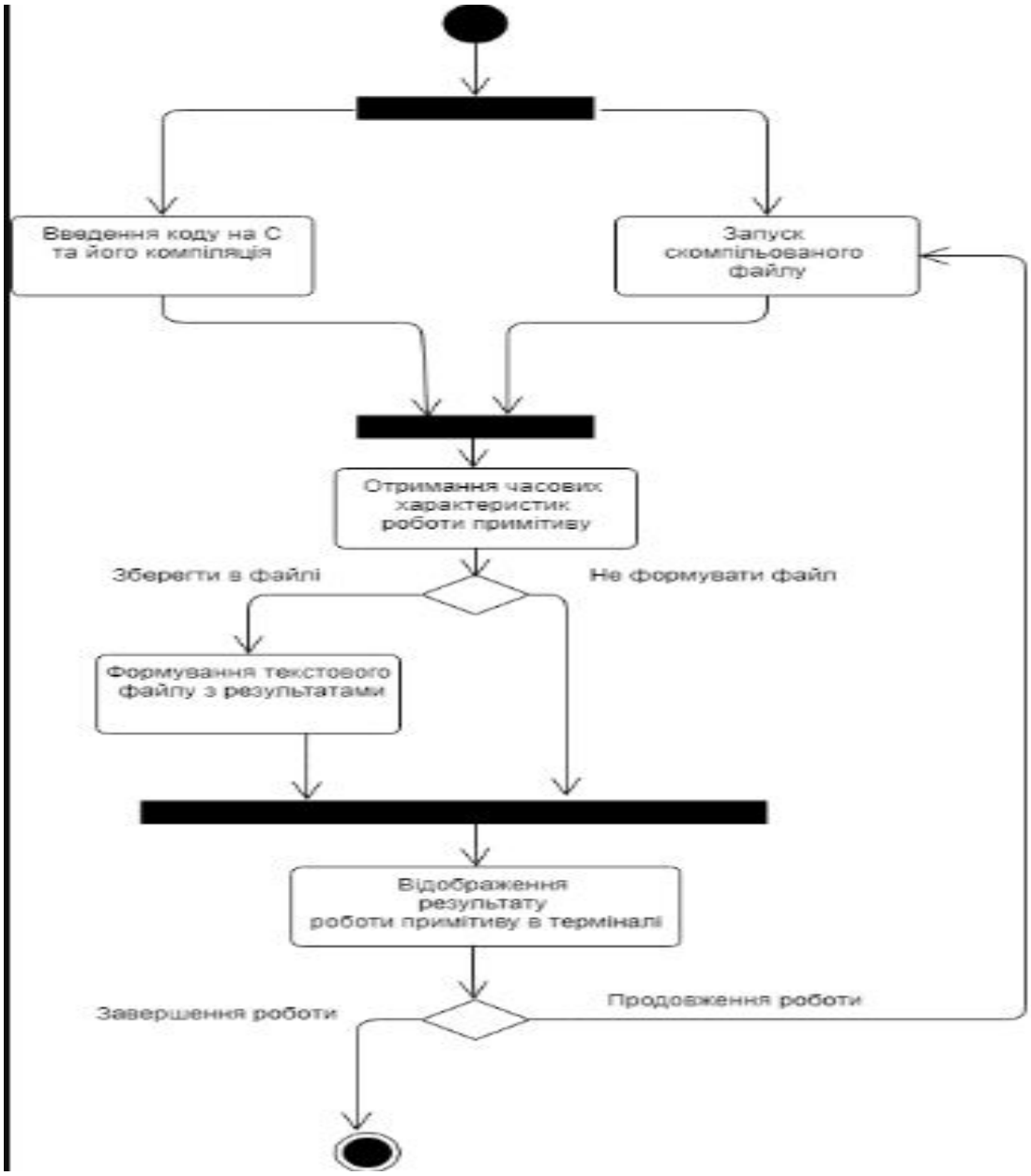


Рисунок 3.3 – Алгоритм послідовності дій програми.

Рисунок 3.4 – Алгоритм послідовності дій програми.

3.2 Використані методи

Програмна система використовує методи часового оцінювання примітивів синхронізації за допомогою спеціальної функції ClockCycles(), що отримує часові характеристики з точністю до наносекунд.

На рис. 3.3 представлена схема взаємодії модулів програми.

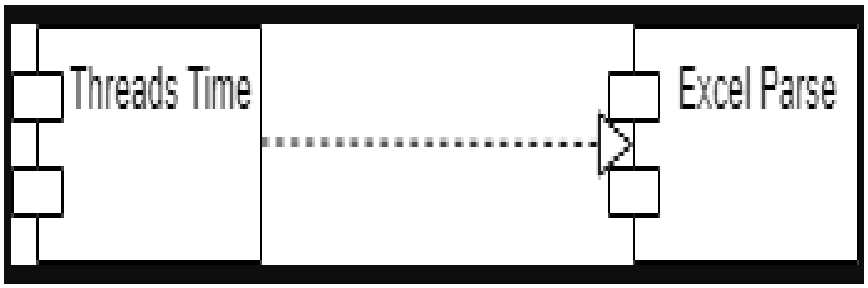


Рисунок 3.3 – Схема взаємодії модулів програми

Стрілками на рис. 3.3 показана залежність одних модулів від інших.

Призначення модулів програми:

– Threads Time – формує текстовий файл з часовими характеристиками роботи примітивів синхронізації;
– Excel Parse – модуль, що здійснює статистичну обробку часових характеристик роботи примітивів синхронізації.

4 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Технічні засоби, які використовуються при роботі програми:

- ПК, під управлінням ОС сімейства QNX;
- процесор з тактовою частотою не нижче 1 ГГц;
- оперативна пам’ять не менше 512 Мб;
- простір на жорсткому диску не менше 512 Мб;
- маніпулятор миша;
- маніпулятор миша;
- наявність CD/DVD приводу, USB роз’єму або LAN-адаптеру для встановлення необхідного ПЗ;
- монітор з роздільною здатністю екрану 1024x768 та більше, з підтримкою не менш ніж 256 кольорів.

5 ВИКЛИК І ЗАВАНТАЖЕННЯ

Перед використанням програми її необхідно запустити або скомпілювати за допомогою терміналу операційної системи:

1) Запустити файл Slp1, Slp2, Slp3, NS1_S, NS2_S, NS3_S, M1_S, M2_S, M3_S, IS1_S, IS2_S, IS3_S, Cond_Var1, Cond_Var2, Cond_Var3 що знаходиться у папці «MasterWork». Це можна зробити за допомогою команди: ./Ім’я файлу.

2) Внести відповідні зміни, та скомпілювати файли Sleepson1_syn.c, Sleepson2_syn.c, Sleepson3_syn.c, NSem1_syn.c, NSem2_syn.c, NSem3_syn.c, Mutex1_syn.c, Mutex2_syn.c, Mutex3_syn.c, ISem1_syn.c, ISem2_syn.c, ISem3_syn.c, Cond_Var1.c, Cond_Var2.c, Cond_Var3.c. Це можна зробити за допомогою команди: gcc ім’я файлу –lstdc++ -o ім’я виконуваного файлу.

3) Отримати часові результати роботи програми.

4) Підтвердити формування текстового файлу з часовими характеристиками роботи програми.

5) Дочекайтесь закінчення роботи та натисніть значок «Х» у вікні терміналу операційної системи.

Після завершення установки, усі необхідні для роботи програми файли будуть розміщені у папці, що була вказана на кроці 1.

Після запуску програма повністю готова до роботи.

6 ВХІДНІ ДАНІ

Вхідними даними програми, що розробляється, є:

- код на мові C/C++, введений вручну в редакторі;
- текстовий файл, який можна перенести в середовище Excel для подальшої обробки;
- коментарі в коді з додатковою інформацію про вхідні дані програми, що аналізується, граничний час виконання окремих функцій.

7 ВИХІДНІ ДАНІ

Вхідними даними програми, що розробляється, є:

- код на мові C/C++, введений вручну в редакторі;
- текстовий файл, який можна перенести в середовище Excel для подальшої обробки;
- коментарі в коді з додатковою інформацію про вхідні дані програми, що аналізується, граничний час виконання окремих функцій.

8 ОПИС ПРИЗНАЧЕНОГО ДЛЯ КОРИСТУВАЧА ІНТЕРФЕЙСУ

Після переходу до папки «QNX C++ SyncPrimComp» з'являється головне вікно програми. На рисунку 8.1 та 8.2 зображено інтерфейс, що надається терміналом операційної системи.

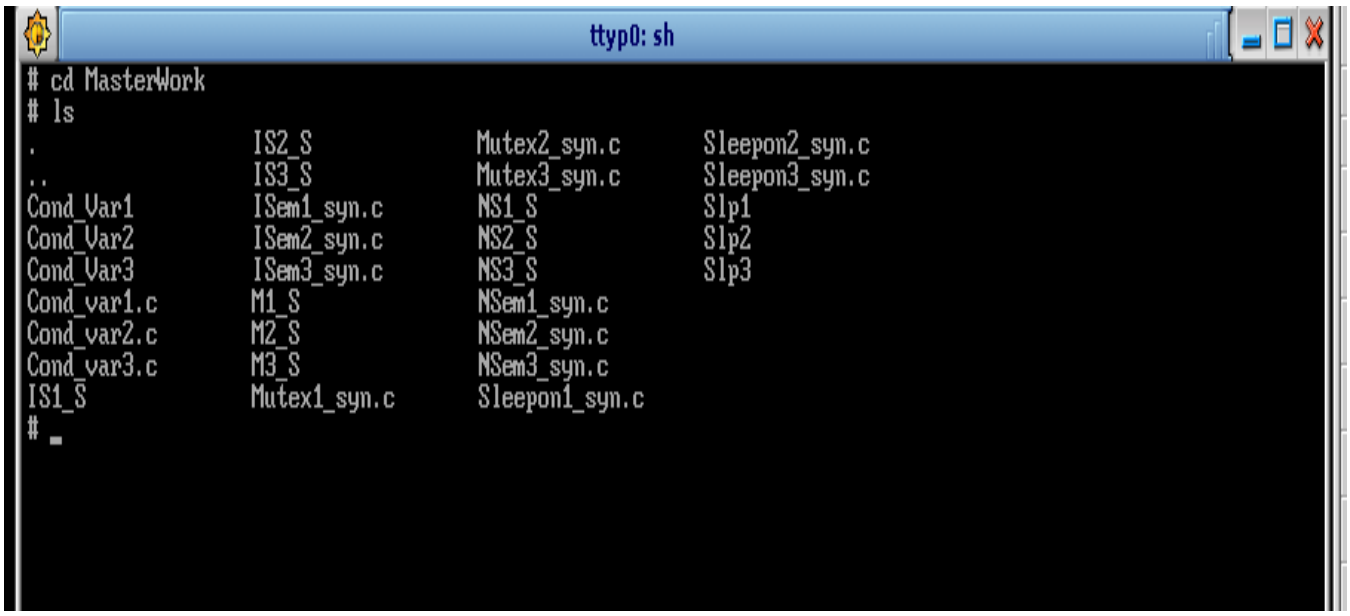
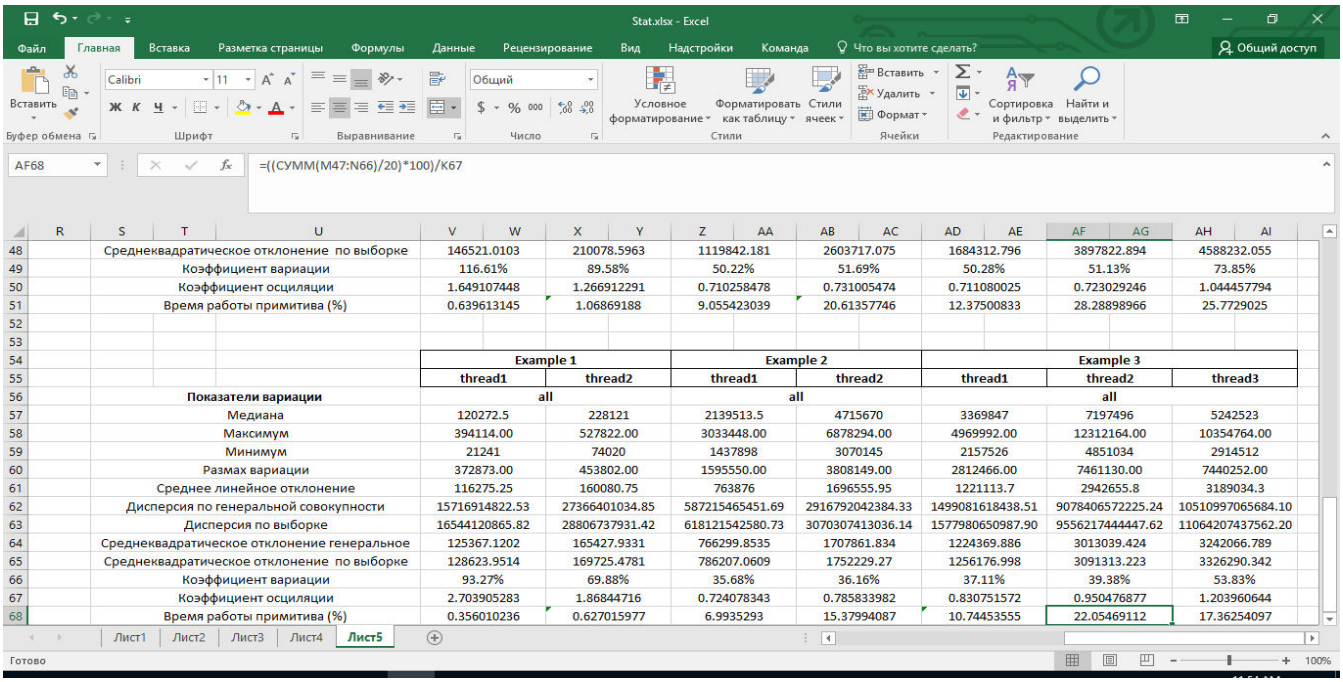


Рисунок 8.1 – Інтерфейс програми



		Example 1		Example 2		Example 3		
		thread1	thread2	thread1	thread2	thread1	thread2	thread3
	Показатели вариации	all		all		all		
48	Среднеквадратическое отклонение по выборке	146521.0103	210078.5963	1119842.181	2603717.075	1684312.796	3897822.894	4588232.055
49	Коэффициент вариации	116.61%	89.58%	50.22%	51.69%	50.28%	51.13%	73.85%
50	Коэффициент осциляции	1.649107448	1.266912291	0.710258478	0.731005474	0.711080025	0.723029246	1.044457794
51	Время работы примитива (%)	0.639613145	1.06869188	9.055423039	20.61357746	12.37500833	28.28898966	25.7729025
57	Медиана	120272.5	228121	2139513.5	4715670	3369847	7197496	5242523
58	Максимум	394114.00	527822.00	3033448.00	6878294.00	4969992.00	12312164.00	10354764.00
59	Минимум	21241	74020	1437898	3070145	2157526	4851034	2914512
60	Размах вариации	372873.00	453802.00	1595550.00	3808149.00	2812466.00	7461130.00	7440252.00
61	Среднее линейное отклонение	116275.25	160080.75	763876	1696555.95	1221113.7	2942655.8	3189034.3
62	Дисперсия по генеральной совокупности	15716914822.53	27366401034.85	587215465451.69	2916792042384.33	1499081618438.51	9078406572225.24	10510997065684.10
63	Дисперсия по выборке	16544120865.82	28806737931.42	618121542580.73	3070307413036.14	1577980650987.90	9556217444447.62	11064207437562.20
64	Среднеквадратическое отклонение генеральное	125367.1202	165427.9331	766299.8535	1707861.834	1224369.886	3013039.424	3242066.789
65	Среднеквадратическое отклонение по выборке	128623.9514	169725.4781	786207.0609	1752229.27	1256176.998	3091313.223	3326290.342
66	Коэффициент вариации	93.27%	69.88%	35.68%	36.16%	37.11%	39.38%	53.83%
67	Коэффициент осциляции	2.703905283	1.86844716	0.724078343	0.785833982	0.830751572	0.950476877	1.203960644
68	Время работы примитива (%)	0.356010236	0.627015977	6.9935293	15.37994087	10.74453555	22.05469112	17.36254097

Рисунок 8.2 – Інтерфейс програми

Головне меню програми складається з наступних пунктів що є базовими для терміналу операційної системи:

– «Зменшити вікно» (відповідає за зменшення);
– «Збільшити вікно» (відповідає за збільшення вікна);
– «Закрити вікно» (відповідає за закриття вікна).

9 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

Перед початком роботи користувач повинен впевнитись в безпечності користуванням ПК або ноутбуком.

Відкрити термінал операційної системи де ввести команду: cd назва папки. В нашому випадку назва папки MasterWork. Ввести системну команду ls для відображення того що знаходиться в даній папці. Для запуску вже скомпільованої програми потрібно ввести в терміналі команду: ./ ім’я файлу. Для роботи з новим файлом його потрібно створити (файл має бути з розширенням .c) в якості прикладів можуть використовуватися файли з розширенням .c що вже реалізовані в даній папці. Після введення і збереження файлу необхідно його скомпілювати в терміналі операційної системи за допомогою команди: gcc ім’я файлу –lstdc++ ім’я виконуваного файлу. Після запуску буде автоматично сформовано і відображено в терміналі операційної системи час роботи примітивів синхронізації операційної системи реального часу QNX, який можна зберегти в текстовий для подальшої роботи з цими даними в середовищі Excel 2016. Після цього можна закрити термінал операційної системи та сформований текстовий файл використати у відповідних полях програми Excel 2016 де буде автоматично виконано підрахунок даних необхідних для коректного порівняння примітивів синхронізації.

10 ПОВІДОМЛЕННЯ

В табл. 10.1 представлені повідомлення користувачу, що можуть з’явитися у процесі роботи програми.

Таблиця 10.1 – Повідомлення, що з’являються в процесі роботи програми

Текст повідомлення	Кому призначене	Опис ситуації	Рекомендовані дії
Примітив синхронізації не створено	Користувач	Виконавчий файл не знайшов в папці файл підключення бібліотеки.	Ввести необхідну бібліотеку та перекомпілювати відповідний файл.
Потік для синхронізації не створено	Користувач	Виконавчий файл не знайшов відповідну бібліотеку або відбулися програмні сбої в системі	Ввести необхідну бібліотеку та перекомпілювати відповідний файл. Або перезавантажити систему.
Помилка ділення на нуль	Користувач	Користувач задав перед компіляцією цикл з лічильником нуль	Внести відповідні зміни в файл з розширенням .c та перекомпілювати його.

ЗАТВЕРДЖЕНО
1116130.01191-01 І3 01-ЛЗ

СИСТЕМА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСОБІВ СИНХРОНІЗАЦІЇ ПОТОКІВ ОС QNX

Керівництво користувача

1116130.01191-01 І3 01

Аркушів 8

2020

ЗМІСТ

ВСТУП

Програмна система «QNX C++ SyncPrimComp» отримує число тактів (періодів частоти), всього демонстраційного прикладу та тої частини програми, яка відповідає за роботу з примітивом синхронізації. Дані формуються в текстовий файл, який передається для обробки в програму Excel, яка підраховує основні статистичні показники (дисперсію, середнє квадратичне відхилення, математичне очікування та графік поведінки кожного з реалізованих примітивів синхронізації, а також час виконання примітиву синхронізації від загального часу роботи програми у %).

Для використання програми, користувачу необхідно вміти працювати в операційній системі реального часу QNX, мати навички користування мишею і клавіатурою, розуміти принцип роботи програми Excel, а також мати знання з мов програмування C/C++.

Перед роботою необхідно ознайомитись з наступними документами:

- опис програми;
- керівництво користувача.

1 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Дана система призначена для побудови графіків, які показують залежність між часом виконання завдання різними примітивами синхронізації та обробки деяких статистичних даних виконання програм, орієнтованих на операційні системи реального часу. Областю застосування програмного продукту може слугувати як сфера програмування в цілому, так і сфера навчання. В сфері програмування, він може використовуватись для визначення оптимального примітиву синхронізації та/або розробки на основі цих даних гібридного примітиву, а в сфері навчання програма може використовуватись викладачами і студентами, наприклад, для порівняння часу роботи примітиву синхронізації двох подібних програм.

Для використання програми необхідна наявність ПК або ноутбуку, на якому встановлена ОС QNX 6 або більш пізня версія. Конфігурація ПК для функціонування даного програмного продукту повинна бути наступна:

- процесор з тактовою частотою не нижче 1 ГГц;
- простір на жорсткому диску не менше 1 ГБ;
- оперативна пам’ять не менше 512 Мб;
- клавіатура;
- наявність CD/DVD приводу, USB роз’єму або LAN-адаптеру для встановлення необхідного ПЗ.

Для використання програми, користувачу необхідно вміти працювати в операційній системі реального часу QNX, знати основи мови програмування C/C++ та ознайомитись з керівництвом користувача.

2 ПІДГОТОВКА ДО РОБОТИ

2.1 Склад і зміст дистрибутивного носія даних

Дистрибутив програмного продукту містить у собі:

- файл a.txt;
- файл CV1,CV2,CV3;
- файл Slp1,Slp2,Slp3;
- файл M1,M2,M3;
- файл NS1,NS2,NS3;
- файл IS1,IS2,IS3.

2.2 Порядок завантаження даних і програм

Для запуску програми необхідно обрати файл, який потрібно запустити, потім відкрити програмний термінал, де необхідно прописати повний шлях до файлу та запустити файл за допомогою команди ./ім’я файлу.

2.3 Порядок перевірки працездатності

Для перевірки працездатності програми необхідно виконати наступний тестовий приклад:

- вибрати необхідний файл;
- відкрити термінал операційної системи;
- прописати повний шлях до програми;
- запустити файл;
- впевнитися, що дані які були отримані є коректними (для цього можна використати сертифіковане програмне забезпечення сторонніх розробників);
- відкрити текстовий файл, який формується за результатом запуску програми;
- отриманий текстовий файл після повторної перевірки можна використати для статистичної обробки в спеціалізованих програмах, зокрема в Excel.

3 ОПИС ОПЕРАЦІЇ

3.1 Функція моделювання роботи примітиву синхронізації

Функція моделює роботу примітиву синхронізації (м’ютекса, іменованого та неіменованого семафору, умовної змінної та сліпсона). Також дана функція підраховує час виконання кожного обробленого примітиву та виводить дану інформацію в термінал операційної системи.

3.2 Функція моделювання задачі

Дана функція моделює задачу, що використовується для отримання різних часових характеристик роботи заданих примітивів синхронізації. В загальному вигляд являє собою цикл в якому залежно від типу задачі змінюється лічильник.

3.3 Функція моделювання кількості потоків

Реалізується в головній частині програми і відповідає за створення та ініціалізацію потоків, а також реалізує лічильник загального часу виконання програми.

4. АВАРІЙНІ СИТУАЦІЇ

За тривалих відмов технічних засобів, що обмежують роботу програми, необхідно звернутись до системного адміністратора.

У випадку збою треба зробити перезавантаження програми або системи. В інших випадках звертатись до системного адміністратора.

У разі відмови магнітних носіїв треба встановити програму з диску на справний магнітний носій.

У разі виявлення помилок у даних потрібно перезапустити програму або систему, в залежності від ступеню важкості збою.

У випадках виявлення несанкціонованого втручання в дані негайно повідомити системного адміністратора про втручання.

В інших аварійних ситуаціях повідомити системного адміністратора про аварійну ситуацію.

ЗАТВЕРДЖЕНО

1116130.01191-01 12 01-ЛЗ

СИСТЕМА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСОБІВ СИНХРОНІЗАЦІЇ ПОТОКІВ В ОС QNX

Текст програми

1116130.01191-01 12 01

Аркушів 18

2020

ЗМІСТ

1 СТРУКТУРА ПРОГРАМИ

Програмний продукт «QNX C++ SyncPrimComp» складається з двох модулів:

– Threads Time – модуль, що відповідає за отримання часових характеристик примітивів синхронізації;

– Excel Parse – модуль, що відповідає за статистичну обробку часових характеристик отриманих примітивів синхронізації.

На рис. 1.1 представлена схема взаємодії модулів програми.

Рисунок 1.1 – Схема взаємодії модулів програми

Модуль ThreadsTime складається з таких файлів написаних на мові С як: Sleepson1_syn.c, Sleepson2_syn.c, Sleepson3_syn.c, NSem1_syn.c, NSem2_syn.c, NSem3_syn.c, Mutex1_syn.c, Mutex2_syn.c, Mutex3_syn.c, ISem1_syn.c, ISem2_syn.c, ISem3_syn.c, Cond_Var1.c, Cond_Var2.c, Cond_Var3.c.

Sleeepon1_syn.c, Sleepon2_syn.c, Sleepon3_syn.c – група С файлів, що отримує часові характеристики роботи примітиву синхронізації неіменований семафор моделюючи різні варіанти виконання цієї програми.

NSem1_syn.c, NSem2_syn.c, NSem3_syn.c – група С файлів, що отримує часові характеристики роботи примітиву синхронізації неіменований семафор моделюючи різні варіанти виконання цієї програми.

Mutex1_syn.c, Mutex2_syn.c, Mutex3_syn.c – група С файлів, що отримує часові характеристики роботи примітиву синхронізації м’ютекс моделюючи різні варіанти виконання цієї програми.

ISem1_syn.c, ISem2_syn.c, ISem3_syn.c – група С файлів, що отримує часові характеристики роботи примітиву синхронізації іменованого семафору моделюючи різні варіанти виконання цієї програми.

Cond_Var1.c, Cond_Var2.c, Cond_Var3.c – група С файлів, що отримує часові характеристики роботи примітиву синхронізації умовної змінної моделюючи різні варіанти виконання цієї програми.

Модуль Excel Parse – являє собою файл формату xlxs, що відповідає за статистичну обробку часових даних роботи примітивів синхронізації отриманих за допомогою модуля Threads Time.

2 ТЕКСТ ПРОГРАМИ

2.1 Cond_Var1.c

```
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <inttypes.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/neutrino.h>

pthread_cond_t cond1=PTHREAD_COND_INITIALIZER;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int done =1;
unsigned long N=3;

 void* foo()
{
unsigned long i;
unsigned long rez=0;
uint64_t t=0, t1;

while(i++!N)
{
t1=ClockCycles();
pthread_mutex_lock(&mutex);
if(done==1) {
done++;
printf(“Done value %d\n”, done);
printf(“Waiting on conditional variable cond1\n”);
pthread_cond_signal(&cond1);
}
else
{
printf(“Signaling conditional variable cond1\n”);
pthread_cond_signal(&cond1);
}
pthread_mutex_unlock(&mutex);
t+=ClockCycles()-t1;
sched_yield();
rez=t/N;
printf(“Returning thread\n”);
printf(“%i cycles %i on cond var “, pthread_self(), t);
printf(“%i\n”, rez);
}
return NULL;
}

int main()
{
pthread_t tid1, tid2;
uint64_t p=0, p1;
p1=ClockCycles();

// Create thread 1
pthread_create(&tid1,NULL,foo,NULL);
// Create thread 2
pthread_create(&tid2,NULL,foo,NULL);

pthread_join(tid1,NULL);
pthread_join(tid2,NULL);
p+=ClockCycles()-p1;

printf(“Program time: %i”, p);
return 0;
}
```

2.2 Cond_Var2.c

```
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <inttypes.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/neutrino.h>

pthread_cond_t cond1=PTHREAD_COND_INITIALIZER;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int done =1;
unsigned long N=3;

void* functionRep()
{
  int num;
  printf("num: %i\n" num);
  for(num=1000000;num>0;i--)
{
}
printf("num: %i\n" num);

}

 void* foo()
{
unsigned long i;
unsigned long rez=0;
uint64_t t=0, t1;

while(i++!N)
{
t1=ClockCycles();
pthread_mutex_lock(&mutex);

if(done==1) {
done++;

functionRep();

printf("Done value %d\n", done);
printf("Waiting on conditional variable cond1\n");
pthread_cond_signal(&cond1);
}
else
{
printf("Signaling conditional variable cond1\n");
pthread_cond_signal(&cond1);
}
pthread_mutex_unlock(&mutex);
t+=ClockCycles()-t1;
sched_yield();
rez=t/N;
printf("Returning thread\n");
printf("%i cycles %i on cond var ", pthread_self(),
t);
printf("%i\n", rez);
}
return NULL;
}

int main()
{
pthread_t tid1, tid2;
uint64_t p=0, p1;
p1=ClockCycles();

// Create thread 1
pthread_create(&tid1,NULL,foo,NULL);
// Create thread 2
pthread_create(&tid2,NULL,foo,NULL);

pthread_join(tid1,NULL);
```

```
pthread_join(tid2,NULL);
p+=ClockCycles()-p1;

printf("Program time: %i", p);
return 0;
}
```

2.3 Cond_Var3.c

```
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <inttypes.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/neutrino.h>

pthread_cond_t cond1=PTHREAD_COND_INITIALIZER;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int done =1;
unsigned long N=3;

void* functionRep()
{
  int num;
  printf("num: %i\n" num);
  for(num=1000000;num>0;i--)
{
}
printf("num: %i\n" num);

}

 void* foo()
{
unsigned long i;
unsigned long rez=0;
uint64_t t=0, t1;

while(i++!N)
{
t1=ClockCycles();
pthread_mutex_lock(&mutex);
if(done==1) {
done++;

functionRep();

printf("Done value %d\n", done);
printf("Waiting on conditional variable cond1\n");
pthread_cond_signal(&cond1);
}
else
{
printf("Signaling conditional variable cond1\n");
pthread_cond_signal(&cond1);
}
pthread_mutex_unlock(&mutex);
t+=ClockCycles()-t1;
sched_yield();
rez=t/N;
printf("Returning thread\n");
printf("%i cycles %i on cond var ", pthread_self(),
t);
printf("%i\n", rez);
}
return NULL;
}

int main()
{
pthread_t tid1, tid2, tid3;
uint64_t p=0, p1;
p1=ClockCycles();
```

```
// Create thread 1
pthread_create(&tid1,NULL,foo,NULL);
// Create thread 2
pthread_create(&tid2,NULL,foo,NULL);
//Create thread3
pthread_create(&tid3,NULL,foo,NULL);



pthread_join(tid1,NULL);
pthread_join(tid2,NULL);
pthread_join(tid3,NULL);
p+=ClockCycles()-p1;

printf("Program time: %i", p);
return 0;
}
```

2.4 ISem1_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <semaphore.h>
#include <fcntl.h>

void* functionC();
static sem_t *sem;
unsigned long N=2;
int counter=0;

int main()
{
int rc1, rc2;
pthread_t thread1, thread2;
int p=0,p1;
const char semname[] ="/duble";


p1=ClockCycles();


if(sem=sem_open(semname,O_CREAT,S_IRWXO,1))==SEM_FA
ILED)
{
perror("semaphore init");
exit(EXIT_FAILURE);
}


if((rc1=pthread_create(&thread1,NULL,&functionC,
NULL)))
{
printf("Thread creation failed: %d\n",rc1);
}


if((rc2=pthread_create(&thread2,NULL,&functionC,
NULL)))
{
printf("Thread creation failed: %d\n",rc2);
}


pthread_join(thread1,NULL);
pthread_join(thread2,NULL);


sem_close(sem);
sem_unlink(semname);


p+=ClockCycles()-p1;
printf("Program time: %i", p);


exit(EXIT_SUCCESS);
}


void* functionC()
{
 long i=0;
```

```
 long rez=0;
int t=0,t1;
while(i++!=N)
{
t1=ClockCycles();
sem_wait(sem);
counter++;
printf("Counter value: %d\n", counter);
sem_post(sem);
t+=ClockCycles()-t1;
//sched_yield();
rez=t/N;
printf("%i cycles: %i on semaphore with name ",
pthread_self(),t);
printf("%i\n", rez);
};
}
```

2.5 ISem2_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <semaphore.h>
#include <fcntl.h>

void* functionC();
void* functionRep();

static sem_t *sem;
unsigned long N=2;
int counter=0;

void* functionRep()
{
int num;
printf("num: %i\n",num);
for(num=1000000;num>0;num--)
{
}
printf("num: %i\n", num);
}

int main()
{
int rc1, rc2;
pthread_t thread1, thread2;
int p=0,p1;
const char semname[] ="/duble";

p1=ClockCycles();

if(sem=sem_open(semname,O_CREAT,S_IRWXO,1))==SEM_FA
ILED)
{
perror("semaphore init");
exit(EXIT_FAILURE);
}

if((rc1=pthread_create(&thread1,NULL,&functionC,
NULL)))
{
printf("Thread creation failed: %d\n",rc1);
}

if((rc2=pthread_create(&thread2,NULL,&functionC,
NULL)))
{
printf("Thread creation failed: %d\n",rc2);
}

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);

sem_close(sem);
```

```
sem_unlink(semname);


p+=ClockCycles()-p1;
printf(“Program time: %i”, p);


exit(EXIT_SUCCESS);
}


void* functionC()
{
 long i=0;
 long rez=0;
int t=0,t1;
while(i++!=N)
{
t1=ClockCycles();
sem_wait(sem);
functionRep();
sem_post(sem);
t+=ClockCycles()-t1;
//sched_yield();
rez=t/N;
printf(“%i cycles: %i on semaphore with name “,
pthread_self(),t);
printf(“%i\n”, rez);
};
}
```

2.6 ISem3_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <semaphore.h>
#include <fcntl.h>


void* functionC();
void* functionRep();


static sem_t *sem;
unsigned long N=2;
int counter=0;


void* functionRep()
{
int num;
printf(“num: %i\n”,num);
for(num=1000000;num>0;num--)
{
}
printf(“num: %i\n”, num);
}


int main()
{
int rc1, rc2, rc3;
pthread_t thread1, thread2, thread3;
int p=0,p1;
const char semname[] =”/duble”;


p1=ClockCycles();


if(sem=sem_open(semname,O_CREAT,S_IRWXO,1))==SEM_FA
ILED)
{
perror(“semaphore init”);
exit(EXIT_FAILURE);
}


if((rc1=pthread_create(&thread1,NULL,&functionC,
NULL)))
{
printf(“Thread creation failed: %d\n”,rc1);
}
```

```
if((rc2=pthread_create(&thread2,NULL,&functionC,
NULL)))
{
printf("Thread creation failed: %d\n",rc2);
}

if((rc3=pthread_create(&thread3,NULL,&functionC,
NULL)))
{
printf("Thread creation failed: %d\n",rc2);
}

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
pthread_join(thread3,NULL);

sem_close(sem);
sem_unlink(semname);

p+=ClockCycles()-p1;
printf("Program time: %i", p);

exit(EXIT_SUCCESS);
}

void* functionC()
{
 long i=0;
 long rez=0;
int t=0,t1;
while(i++!=N)
{
t1=ClockCycles();
sem_wait(sem);
functionRep();
sem_post(sem);
t+=ClockCycles()-t1;
//sched_yield();
rez=t/N;
printf("%i cycles: %i on semaphore with name ",
pthread_self(),t);
printf("%i\n", rez);
};
}
```

2.7 Mutex1_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>

void* functionC();
pthread_mutex_t mutex1= PTHREAD_MUTEX_INITIALIZER;
unsigned long N=2;
int counter =0;

int main()
{
int rc1,rc2;
pthread_t thread1,thread2;
uint64_t p=0,p1;

p1=ClockCycles();
if((rc1=pthread_create(
&thread1,NULL,&functionC,NULL)))
{
printf("Thread creation failed: %d\n",rc1);
}

if((rc2=pthread_create(
&thread2,NULL,&functionC,NULL)))
{
printf("Thread creation failed: %d\n",rc2);
```

```
}

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
p+=ClockCycles()-p1;
printf("Program time: %i\n",p);

exit(EXIT_SUCCESS);
}

void* functionC()
{
unsigned long i=0;
unsigned long rez=0;
uint64_t t=0,t1;
while(i++!=N)
{
t1=ClockCycles();
pthread_mutex_lock(&mutex1);
counter++;
printf("Counter value: %d\n", counter);
pthread_mutex_unlock(&mutex1);
t+=ClockCycles()-t1;
//sched_yield();
rez=t/N;
printf("%i cycles %i on mutex ", pthread_self(),
t);
printf("%i\n", rez);

};
}
```

2.8 Mutex2_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>

void* functionC();
void* functionRep();
pthread_mutex_t mutex1= PTHREAD_MUTEX_INITIALIZER;
unsigned long N=2;
int counter =0;

void* functionRep()
{
int num;
printf("num: %i\n", num);
for(num=1000000;num>0;num--)
{
}

printf("num: %i\n", num);
}

int main()
{
int rc1,rc2;
pthread_t thread1,thread2;
uint64_t p=0,p1;

p1=ClockCycles();
if((rc1=pthread_create(
&thread1,NULL,&functionC,NULL)))
{
printf("Thread creation failed: %d\n",rc1);
}

if((rc2=pthread_create(
&thread2,NULL,&functionC,NULL)))
{
printf("Thread creation failed: %d\n",rc2);
}
```

```
pthread_join(thread1,NULL);

pthread_join(thread2,NULL);

p+=ClockCycles()-p1;

printf("Program time: %i\n",p);


exit(EXIT_SUCCESS);

}


void* functionC()

{
unsigned long i=0;

unsigned long rez=0;

uint64_t t=0,t1;

while(i++!=N)

{

t1=ClockCycles();

pthread_mutex_lock(&mutex1);

functionRep();

pthread_mutex_unlock(&mutex1);

t+=ClockCycles()-t1;

//sched_yield();

rez=t/N;

printf("%i cycles %i on mutex ", pthread_self(),
t);

printf("%i\n", rez);


};

}
```

2.9 Mutex3_syn.c

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <inttypes.h>

#include <unistd.h>

#include <errno.h>

#include <sys/neutrino.h>


void* functionC();

void* functionRep();

pthread_mutex_t mutex1= PTHREAD_MUTEX_INITIALIZER;

unsigned long N=2;

int counter =0;


void* functionRep()

{

int num;

printf("num: %i\n", num);

for(num=1000000;num>0;num--)

{

}


printf("num: %i\n", num);

}



int main()

{

int rc1,rc2,rc3;

pthread_t thread1,thread2,thread3;

uint64_t p=0,p1;


p1=ClockCycles();

if((rc1=pthread_create(
&thread1,NULL,&functionC,NULL)))

{

printf("Thread creation failed: %d\n",rc1);

}


if((rc2=pthread_create(
&thread2,NULL,&functionC,NULL)))

{

printf("Thread creation failed: %d\n",rc2);

}
```

```
if((rc3=pthread_create(
&thread3,NULL,&functionC,NULL)))
{
printf("Thread creation failed: %d\n",rc3);
}

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
pthread_join(thread3,NULL);

p+=ClockCycles()-p1;
printf("Program time: %i\n",p);

exit(EXIT_SUCCESS);
}

void* functionC()
{
unsigned long i=0;
unsigned long rez=0;
uint64_t t=0,t1;
while(i++!=N)
{
t1=ClockCycles();
pthread_mutex_lock(&mutex1);
functionRep();
pthread_mutex_unlock(&mutex1);
t+=ClockCycles()-t1;
//sched_yield();
rez=t/N;
printf("%i cycles %i on mutex ", pthread_self(),
t);
printf("%i\n", rez);

};
}
```

2.10 NSem1_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <semaphore.h>

void* functionC();
static sem_t sem;
unsigned long N=2;
int counter =0;

int main()
{
int rc1,rc2;
pthread_t thread1,thread2;
uint64_t p=0,p1;

p1=ClockCycles();

if(sem_init(&sem,0,1)!=0)
{
perror("semaphore init");
exit(EXIT_FAILURE);
}

if((rc1=pthread_create(
&thread1,NULL,&functionC,NULL)))
{
printf("Thread creation failed: %d\n",rc1);
}

if((rc2=pthread_create(
&thread2,NULL,&functionC,NULL)))
```

```
{
printf("Thread creation failed: %d\n",rc2);
}

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);

sem_destroy(&sem);

p+=ClockCycles()-p1;
printf("Program time: %i\n",p);

exit(EXIT_SUCCESS);
}

void* functionC()
{
unsigned long i=0;
unsigned long rez=0;

uint64_t t=0,t1;
while(i++!=N)
{
t1=ClockCycles();
sem_wait(&sem);
counter++;
printf("Counter value: %d\n", counter);
sem_post(&sem);
t+=ClockCycles()-t1;
//sched_yield();
rez=t/N;
printf("%i cycles %i on semaphore (without name)",
pthread_self(), t);
printf("%i\n", rez);
};
}
```

2.11 NSem2_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <semaphore.h>

void* functionC();
void* functionRep();
static sem_t sem;
unsigned long N=2;
int counter =0;

void* functionRep()
{
int num;
printf("num: %i\n", num);
for(num=1000000;num>0;num--)
{
}
printf("num: %i\n", num);
}

int main()
{
int rc1,rc2;
pthread_t thread1,thread2;
uint64_t p=0,p1;

p1=ClockCycles();
```

```
if(sem_init(&sem,0,1)!=0)
{
perror("semaphore init");
exit(EXIT_FAILURE);
}

if((rc1=pthread_create(
&thread1,NULL,&functionC,NULL)))
{
printf("Thread creation failed: %d\n",rc1);
}

if((rc2=pthread_create(
&thread2,NULL,&functionC,NULL)))
{
printf("Thread creation failed: %d\n",rc2);
}

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);

sem_destroy(&sem);

p+=ClockCycles()-p1;
printf("Program time: %i\n",p);

exit(EXIT_SUCCESS);
}

void* functionC()
{
unsigned long i=0;
unsigned long rez=0;

uint64_t t=0,t1;
while(i++!=N)
{
t1=ClockCycles();
sem_wait(&sem);
functionRep();
sem_post(&sem);
t+=ClockCycles()-t1;
//sched_yield();
rez=t/N;
printf("%i cycles %i on semaphore (without name)",
pthread_self(), t);
printf("%i\n", rez);
};
}
```

2.12 NSem3_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <semaphore.h>

void* functionC();
void* functionRep();
static sem_t sem;
unsigned long N=2;
int counter =0;

void* functionRep()
{
int num;
printf("num: %i\n", num);
for(num=1000000;num>0;num--)
{
}
printf("num: %i\n", num);
}
```

```
int main()
{
int rc1,rc2,rc3;
pthread_t thread1,thread2, thread3;
uint64_t p=0,p1;

p1=ClockCycles();



if(sem_init(&sem,0,1)!=0)
{
perror(“semaphore init”);
exit(EXIT_FAILURE);
}

if((rc1=pthread_create(
&thread1,NULL,&functionC,NULL)))
{
printf(“Thread creation failed: %d\n”,rc1);
}

if((rc2=pthread_create(
&thread2,NULL,&functionC,NULL)))
{
printf(“Thread creation failed: %d\n”,rc2);
}

if((rc3=pthread_create(
&thread3,NULL,&functionC,NULL)))
{
printf(“Thread creation failed: %d\n”,rc3);
}

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
pthread_join(thread3,NULL);

sem_destroy(&sem);

p+=ClockCycles()-p1;
printf(“Program time: %i\n”,p);

exit(EXIT_SUCCESS);
}

void* functionC()
{
unsigned long i=0;
unsigned long rez=0;

uint64_t t=0,t1;
while(i++!=N)
{
t1=ClockCycles();
sem_wait(&sem);
functionRep();
sem_post(&sem);
t+=ClockCycles()-t1;
//sched_yield();
rez=t/N;
printf(“%i cycles %i on semaphore (without name)”,
pthread_self(), t);
printf(“%i\n”, rez);
};
}
```

2.13 Sleepon1_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>
```

```
int done =1;
unsigned long N=2;


void* foo()
{
int dr;
unsigned long i;
unsigned long rez=0;
uint64_t t=0, t1;

while(i++!=N)
{
t1=ClockCycles();
pthread_sleepon_lock();
if(done==1)
{
done++;
printf("Waiting on sleepon1\n");
pthread_sleepon_wait(&done);
printf("Done value: %d\n", done);
done=0;
}
else
{
done=1;
printf("Signaling sleepon1\n");
pthread_sleepon_signal(&done);
}
pthread_sleepon_unlock();
t+=ClockCycles()-t1;
sched_yield();
rez=t/N;
printf("Returning thread\n);
printf("%i cycles: %i on sleepon ",
pthread_self(),t);
printf("%i\n", rez);
}

return NULL;
}

int main()
{
pthread_t tid1,tid2;

uint64_t p=0,p1;
p1=ClockCycles();

// Create thread 1
pthread_create(&tid1,NULL,foo,NULL);
// Create thread2
pthread_create(&tid2,NULL,foo,NULL);


pthread_join(tid1,NULL);
pthread_join(tid2,NULL);

p+=ClockCycles()-p1;
printf("\nProgram work: %i\n",p);
return 0;
}
```

2.14 Sleepon2_syn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>

int done =1;
```

```
unsigned long N=2;


void* functionRep()
{
int num;
printf("num: %i\n", num);
for(num=1000000;num>0;num--)
{
}
printf("num: %i\n", num);
}


void* foo()
{
int dr;
unsigned long i;
unsigned long rez=0;
uint64_t t=0, t1;


while(i++!=N)
{
t1=ClockCycles ();
pthread_sleepon_lock();
if(done==1)
{
functionRep();
printf("Waiting on sleepon1\n");
pthread_sleepon_wait(&done);
printf("Done value: %d\n", done);
done=0;
}
else
{
done=1;
printf("Signaling sleepon1\n");
pthread_sleepon_signal(&done);
}
pthread_sleepon_unlock();
t+=ClockCycles()-t1;
sched_yield();
rez=t/N;
printf("Returning thread\n);
printf("%i cycles: %i on sleepon ",
pthread_self(),t);
printf("%i\n", rez);


}


return NULL;
}


int main()
{
pthread_t tid1,tid2;


uint64_t p=0,p1;
p1=ClockCycles();


// Create thread 1
pthread_create(&tid1,NULL,foo,NULL);
// Create thread2
pthread_create(&tid2,NULL,foo,NULL);



pthread_join(tid1,NULL);
pthread_join(tid2,NULL);


p+=ClockCycles()-p1;
printf("\nProgram work: %i\n",p);
return 0;
}
```

2.15 Sleepon3_syn.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <pthread.h>
#include <string.h>
#include <inttypes.h>
#include <unistd.h>
#include <errno.h>
#include <sys/neutrino.h>

int done =1;
unsigned long N=2;

void* functionRep()
{
int num;
printf("num: %i\n", num);
for(num=1000000;num>0;num--)
{
}
printf("num: %i\n", num);
}

void* foo()
{
int dr;
unsigned long i;
unsigned long rez=0;
uint64_t t=0, t1;

while(i++!=N)
{
t1=ClockCycles ();
pthread_sleepon_lock();
if(done==1)
{
functionRep();
printf("Waiting on sleepon1\n");
pthread_sleepon_wait(&done);
printf("Done value: %d\n", done);
done=0;
}
else
{
done=1;
printf("Signaling sleepon1\n");
pthread_sleepon_signal(&done);
}
pthread_sleepon_unlock();
t+=ClockCycles()-t1;
sched_yield();
rez=t/N;
printf("Returning thread\n);
printf("%i cycles: %i on sleepon ",
pthread_self(),t);
printf("%i\n", rez);

}

return NULL;
}

int main()
{
pthread_t tid1,tid2,tid3;

uint64_t p=0,p1;
p1=ClockCycles();

// Create thread 1
pthread_create(&tid1,NULL,foo,NULL);
// Create thread2
pthread_create(&tid2,NULL,foo,NULL);
// Create thread3
pthread_create(&tid3,NULL,foo,NULL);

pthread_join(tid1,NULL);
pthread_join(tid2,NULL);
pthread_join(tid3,NULL);
```

```
p+=ClockCycles()-p1;

printf(“\nProgram work: %i\n”,p);

return 0;

}
```

Оцінка впливу випадкових характеристик обчислювальних процесів на найгірший час виконання програм (Worst Case Execution Time, WCET)

Нечай В. Я., Кочерга С. А., Сєнін Д. С. Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

На даний час у літературі розглядаються та досліджуються десятки різних вимірів продуктивності виконання прикладних програм. Але всі існуючі тлумачення вимірів продуктивності так чи інакше пов'язані з часовими характеристиками системи.

Для систем реального часу поряд із продуктивністю актуальною є також їхня передбачуваність. Цей критерій також пов'язаний з часовими характеристиками та означає, що такі показники, як час виконання системних викликів, час затримки переривань, час маскування переривань, вплив засобів синхронізації потоків та інші, повинні бути оцінені заздалегідь з певною точністю. Ці показники є складовими найгіршого часу виконання програм (WCET) та мають випадковий характер.

У роботі [Нечай, Волошин, Нежуміра, 2017] були виконані дослідження з оцінки WCET за допомогою статистичних, аналітичних та гібридних методів і надана їх оцінка. Рекомендовано для надійності отримані результати збільшити на 15-20%. Це гарантує, що WCET не перевищуватиме «реальний час», який визначається фізичними процесами. Але, з іншого боку, така груба оцінка WCET може призвести до невиправданого завищення витрат ресурсів системи. Тому доцільно досліджувати ступінь впливу випадкових показників на величину WCET з метою його можливого уточнення. Видається очевидним, що ступінь впливу цих випадкових показників буде залежати від тривалості часового інтервалу, що досліджується. Враховуючи, що час перебігу деяких процесів (наприклад, при управлінні хімічними реакціями) достатньо малий, вплив цих випадкових складових для подібних задач може виявитися вельми істотним. Більше того, можна очікувати, що цей час буде порівнюваним з постійною складовою WCET. Тому оцінка впливу окремих випадкових складових для певного класу задач може уточнити оцінку WCET.

Система реального часу QNX має багатий набір функцій для дослідження часових характеристик. Вона містить у собі як функції стандарту POSIX, ANSI та UNIX, так і функції, притаманні тільки службі часу QNX. Вимоги, які пред'являються до служби часу операційної системи (ОС) QNX, значно ширші, ніж для ОС загального призначення. Так, користувачеві надаються можливості реалізації програмних таймерів із завданням проміжків часу від десятків наносекунд до декількох років. Крім того, є можливість регулювати роздільну здатність системного годинника, що дозволить зменшити похибки, що привносяться такими явищами, як затримка переривань, флуктуація відліку часу та іншими.

Отримані результати дозволять виділити клас задач, для яких вказані випадкові показники є суттєвими, а також задачі, де ними можна знехтувати.