ISSN 2074-7888, Наукові праці Донецького національного технічного університету, серія "Проблеми моделювання та автоматизації проектування" №1 (14), 2019

УДК 004.45 DOI: 10.31474/2074-7888-2019-1-43-52

# УПРАВЛІННЯ ПАМ'ЯТТЮ І СТРУКТУРИ ДАНИХ В ОПЕРАЦІЙНИХ СИСТЕМАХ FREERTOS ТА PREDICATE OS

Гайдук К. С., Шевченко О. Г.

*В роботі наведено результати дослідження та порівняльного аналізу алгоритмів динамічного розподілу пам'яті та базових структур даних, що використовуються у вбудованих операційних системах FreeRTOS і Predicate OS. Запропоновано поняття умовної верхньої межі часу виконання операції та коефіцієнту детермінованості, що дозволяють виконувати кількісну оцінку детермінованості роботи алгоритму. Дослідження виконувалися на базі платформи STM32 із застосуванням таймера високої роздільної здатності DWT. Отримані результати можуть бути корисними для розробників програмного забезпечення при виборі алгоритмів динамічного розподілу пам'яті та їх параметрів.*

***Ключові слова:*** *алгоритми динамічного розподілу пам'яті, детермінізм алгоритму, динамічні списки, черги, накладні витрати ресурсів, операційні системи реального часу.*

**Вступ**. Використання динамічного розподілу пам'яті (ДРП) у вбудованих системах реального часу (СРЧ) є обмеженим, оскільки знижує часовий детермінізм системи, обумовлює накладні витрати оперативної пам'яті та процесорного часу, збільшує час відгуку [1,2]. Проте, ДРП все частіше застосовується у вбудованих СРЧ, на що є кілька причин: 1 – поява алгоритмів ДРП зі сталим часом виконання; 2 – поява досить потужних апаратних платформ для вбудованих систем; 3 – використання ДРП наділяє програмне забезпечення більшою гнучкістю, ніж у разі статичного розподілу пам'яті. Прикладами операційних систем реального часу (ОСРЧ), що використовують ДРП, можуть слугувати такі, як RT-Linux GPL, QNX Neutrino, MARTE OS [1] та ін.

Окрім збільшення гнучкості програмного забезпечення, ДРП також опосередковано дозволяє досягти запобігання ресурсних взаємних блокувань [3, 4], що було використано в авторській операційній системі Predicate OS. Одним з ключових архітектурних елементів зазначеної ОС є поштові скриньки задач (черги повідомлень), реалізовані на базі динамічних списків, що робить розподільник пам'яті та черги повідомлень найбільш вузьким місцем системи в плані витрат процесорного часу і пам'яті. Для оцінки життєздатності використаної архітектури було прийнято рішення виконати порівняння Predicate OS з такою популярною вбудованою операційною системою реального часу, як FreeRTOS [5].

**Аналіз останніх досліджень та публікацій**. Існує велика кількість алгоритмів ДРП [6-12], серед яких наявні як ті, що міцно увійшли в інженерну практику, так і ті, що представляють виключно теоретичний інтерес. Серед усіх алгоритмів ДРП лише деякі характеризуються сталим часом виділення та звільнення блоку пам'яті (прикладами можуть слугувати TLSF та Half-Fit), решта ж має час $O(n)$ для обох, або однієї з вказаних операцій [6, 8, 9, 11]. В [10] доводиться детермінованість за часом алгоритму DL malloc, який застосовується в багатьох операційних системах і реалізаціях стандартної бібліотеки stdlib мови програмування C, хоча він не знайшов широкого застосування в ОСРЧ.

Операційна система FreeRTOS надає п'ять схем розподілу пам'яті [13], перша з яких дозволяє лише виділяти пам'ять, і не підтримує функції звільнення блоків пам'яті. Інші чотири представлені алгоритмами Best-Fit, First-Fit, розподільником зі стандартної бібліотеки stdlib, а також модифікацією First-Fit, що дозволяє організовувати складову купу на базі кількох не суміжних областей пам'яті. Незважаючи на те, що FreeRTOS позиціонується як ОСРЧ, застосовані в ній алгоритми Best-Fit і First-Fit мають найгірший час виділення блоку, рівний $O(n)$, де під параметром $n$ розуміється довжина списку вільних блоків.

Раніше згаданий алгоритм TLSF, використаний в Predicate OS, не підтримує функції динамічної зміни розміру купи sbrk(), однак це обмеження не представляється критичним для переважної більшості вбудованих систем.

**Метою** даної статті є дослідження і порівняльний аналіз алгоритмів динамічного розподілу пам'яті, а також базових структур даних, що використовуються в операційних системах FreeRTOS та Predicate OS.

**Матеріали та методи.** Для кількісної оцінки та порівняння алгоритмів ДРП використовувалися наступні показники: мінімальне, середнє і максимальне значення часу виконання операції; стандартне відхилення на основі зміщеної оцінки дисперсії; коефіцієнт коваріації. У разі дослідження операції виділення пам'яті в різних алгоритмах, при розрахунку вищезазначених показників бралися до уваги лише випадки вдалого виконання операції (функція повертає не нульовий покажчик).

Незважаючи на те, що перелічені показники надають деяку характеристику алгоритму, вони є малоінформативними в контексті оцінки детермінізму, тому як в останньому випадку найбільший інтерес представляє розкид випадкової величини відносно деякої верхньої межі, а не відносно середнього значення. У зв'язку з цим в роботі введено поняття умовної верхньої межі часу виконання операції, а також коефіцієнта детермінованості, смисл і алгоритм розрахунку яких наведено нижче.

Нехай дано $N$ результатів вимірювання часу виконання деякої операції. Множина даних відліків ділиться на рівні інтервали по $d$ відліків в кожному (1):

$$d = \lfloor N/k \rfloor, \quad (1)$$

де $k$ має порядок близько 10.

Очевидно, що кількість отриманих інтервалів $m$ буде визначатися виразом (2):

$$m = \begin{cases} \lfloor N/k \rfloor, if\ N\ mod\ k = 0 \\ \lfloor N/k \rfloor + 1, if\ N\ mod\ k \neq 0, \end{cases} \quad (2)$$

Для кожного інтервалу знаходиться максимальне значення, що дає $m$ локальних максимумів $LM_i$. Середнє арифметичне всіх локальних максимумів назвемо умовною верхньою межею часу виконання $Th$ (3):

$$Th = \frac{1}{m}\sum_{i=1}^{m} LM_i. \quad (3)$$

Коефіцієнт детермінованості $K_d$ розраховується як середнє квадратичне різниць $t_i - Th$ (4), де $t_i$ - $i$-й відлік. При цьому враховуються лише ті точки, що лежать вище $Th$ (рис. 1):

$$K_d = \frac{1}{r}\sum_{i=1}^{N}(t_i - Th)^2, if\ t_i - Th > 0, \quad (4)$$

де $r$ – кількість точок, для яких виконується умова $t_i - Th > 0$.

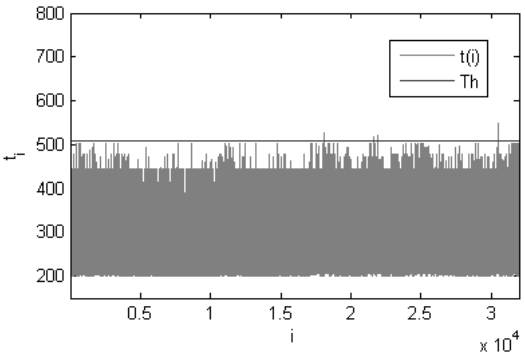


Рис. 1 – Умовна верхня межа часу виконання операції.

Для отримання кількісних характеристик різних алгоритмів ДРП було використано дві схеми проведення експериментів:

*Схема 1*. В межах одного кроку циклу виконується вісім запитів на виділення блоку пам'яті і стільки ж операцій звільнення. Порядок запитів на виділення і звільнення блоку пам'яті перемішаний [14].

Кількість ітерацій циклу у всіх експериментах постійна і становить 8000. Розмір купи $H_s$ у всіх випадках дорівнює 15 Кб. Діапазон розмірів запитуваних блоків дорівнює $[0,..,H_s/Q-1]$, де $Q$ - параметр, який приймає значення 1 або 20, що відповідає двом окремим варіантам експерименту.

Функцію-перехоплювач невдалого виділення пам'яті в FreeRTOS [2] відключено.

*Схема 2*. На кожній ітерації циклу виконується або випадкова кількість запитів на виділення блоку пам'яті, або випадкова кількість запитів на звільнення. Причому, запити на виділення виконуються частіше, ніж на звільнення [14]. Кількість ітерацій дорівнює 64000. Як і у разі першої схеми, проводяться окремі експерименти для різних значень параметра $Q$. В обох схемах ведеться підрахунок кількості відмов.

З метою більш ретельного аналізу деяких алгоритмів, використані спеціальні вставки у вихідному коді, що дозволяють

досягти «декомпозиції» одержуваних графіків і виконати їх інтерпретацію (рис. 7-8).

Операції виділення блоку пам'яті (malloc()) і звільнення (free()) є нереєнтерабельними, і тому основний код відповідних функцій має виконуватися всередині критичних секцій (КС). Незважаючи на те, що обробники завдань-автоматів в Predicate OS [3] є атомарними і не можуть переривати виконання один одного, вони можуть перериватися обробниками апаратних переривань, тому організація КС актуальна також і для даної операційної системи.

З метою дослідження роботи розподільників із обробників апаратних переривань, використовувалися модифіковані варіанти вищеописаних схем: паралельно з циклом в основній програмі, що містить запити malloc() і free(), ті ж запити по черзі надходили також з обробника одного з апаратних таймерів системи [14].

*Списки і черги*. Як було зазначено раніше, в Predicate OS черги реалізовано на базі двозв'язних динамічних списків. Кожен елемент списку має наступні поля: покажчик на довільні дані; покажчики на попередній і наступний елементи; пріоритет (можлива організація черг з пріоритетами); крайній термін обробки; код повідомлення (події). Останні два поля актуальні для поштових скриньок задач. При створенні списку, вказується максимальна кількість елементів, яку він може містити (для створення списку з необмеженою кількістю елементів, необхідно вказати константу MAX_LIST_SIZE).

З тієї причини, що списки використовуються для організації не тільки черг без пріоритетів, а й черг з пріоритетами, створення списків на базі масивів (без виділення окремого блоку пам'яті під кожен елемент списку) є складним, і нераціональним в плані накладних витрат процесорного часу.

В FreeRTOS черги є безпріорітетними, і організовані на базі масивів. Черги суміщені із засобами синхронізації (двійкові семафори, рахункові семафори, м'ютекси або рекурсивні м'ютекси). Таким чином, черга може перебувати в блокованому або вільному стані. Ряд функцій для роботи з чергами мають параметр, що задає таймаут для виконання операції. В Predicate OS подібні механізми синхронізації не потрібні, через використання парадигми автоматного програмування [3, 15].

FreeRTOS не надає API для роботи зі списками [16], хоча використовує двозв'язний сортований (за спаданням) динамічний список при плануванні виконання задач.

Оцінка асимптотичної складності алгоритмів за часом, а також накладних витрат пам'яті, обумовлених використанням алгоритмами різних службових структур даних, виконувалася на підставі аналізу відповідних вихідних кодів. У всіх експериментах, випробування проводилися за умови відсутності в системі будь-яких задач, що забезпечує коректність одержуваних вимірів часу.

Робота виконана на базі апаратної платформи мікроконтролера STM32F103C8, що має ядро Cortex-M3, 20 Кб SRAM та 64 Кб FLASH. Тактова частота - 8 МГц. Вимірювання часових інтервалів здійснювалося за допомогою таймера високої роздільної здатності DWT (Data Watchpoint and Trace unit). Передача результатів експерименту на хост виконувалася за допомогою інтерфейсу USART. Версії операційних систем: FreeRTOS 10 і Predicate OS 1.1.

**Результати та їх обговорення.** Результати дослідження алгоритмів ДРП наведено в табл. 1-4 та на рис. 2-5. Було розглянуто чотири схеми розподілу пам’яті, що використовуються в операційній системі FreeRTOS (heap2.c-heap5.c), а також дві реалізації алгоритму TLSF, застосованого в Predicate OS: реалізація виключно на мові С, та із застосуванням асемблерних вставок (команди CLZ та RBIT з системи команд Cortex-M3). Позначення, використовувані в таблицях: $F$ – відсоток відмов при запитах на виділення блоку пам’яті, $T_{min}$, $T_{max}$, $T_{\text{сер}}$ – відповідно, мінімальний, максимальний та середній час виконання операції (в тактах процесору), $S$ - стандартне відхилення на основі зміщеної оцінки дисперсії, $K$ – коефіцієнт коваріації, $Th$ – умовна верхня межа часу виконання, $K_d$ – коефіцієнт детермінованості.

З отриманих результатів видно, що алгоритм Best-Fit характеризується найбільшим часом виконання (серед розглянутих алгоритмів), та найгіршим детермінізмом. Час виконання операцій виділення та звільнення пам’яті для цього алгоритму пропорційний фрагментації купи (рис. 6).

Розкид часу виконання операції malloc() для алгоритмів DL malloc та First-Fit на рисунку 2 менше, ніж на рисунках 3-4, що

обумовлено відмінністю у схемах проведення експериментів: у разі першої схеми, на початок кожного кроку моделювання купа знаходиться в нефрагментованому стані. Кількість запитів malloc() фіксована, і це обумовлює обмеженість розміру структур даних (списків, дерев) для кожного з зазначених алгоритмів. У разі використання другої схеми, розміри структур даних не детерміновані алгоритмом моделювання, і розкид часу виконання операції помітно збільшується.

Як у разі імплементації алгоритму TLSF виключно мовою C, так і в разі використання асемблерних вставок, найгірший час виконання операцій malloc() і free() дорівнює O(1), однак різниця між даними двома варіантами реалізації очевидна (рис. 2-5). Якщо для TLSF (C) коефіцієнт детермінованості $K_d$ складає 1.62 (табл. 3), то для TLSF (C + ASM) $K_d = 0$. Алгоритм TLSF дещо поступається алгоритмам DL malloc та First-Fit за середнім часом виконання (табл. 3), однак має практично ідеальний детермінізм, у разі використання асемблерних вставок ($K_d = 0$ проти $K_d = 15.75$ та $K_d = 0.83$ для DL malloc та First-Fit відповідно). Умовна верхня межа часу виконання $Th$ операції malloc() для алгоритмів TLSF та First-Fit приблизно однакова: 373 та 359 відповідно (табл. 3).

Також помітна різниця у результатах експериментів для випадків $Q = 1$ (запитувані розміри блоків належать діапазонові [0; 15359]) та $Q = 20$ (запитувані розміри блоків належать діапазонові [0; 767]): кількість відмов у разі $Q = 20$ значно нижча, ніж для $Q = 1$. Так, при $Q = 1$ та другій схемі, відсоток відмов для всіх алгоритмів становить не менше 36%. У разі ж $Q = 20$, $F$ зменшується майже до нуля, і лише в алгоритму Best-Fit лишається рівною ~38%.

Табл. 1 – Дослідження алгоритмів ДРП за першою схемою ($Q = 1$, операція виділення пам’яті).

| № | Алгоритм (схема, реалізація) | $F, \%$ | $T_{min}$ | $T_{max}$ | $T_{\text{сер}}$ | $S$ | $K, \%$ |
|---|---|---|---|---|---|---|---|
| 1 | Best-Fit (heap2.c) | 97 | 180 | 2848 | 1685 | 500 | 30 |
| 2 | DL malloc (heap3.c) | 46 | 197 | 571 | 241 | 89 | 37 |
| 3 | First-Fit (heap4.c) | 46 | 201 | 299 | 253 | 9 | 4 |
| 4 | First-Fit (heap5.c) | 53 | 195 | 308 | 251 | 18 | 7 |
| 5 | TLSF (C) | 47 | 287 | 783 | 583 | 74 | 13 |
| 6 | TLSF (C + ASM) | 47 | 222 | 373 | 358 | 25 | 7 |

Табл. 2 – Дослідження алгоритмів ДРП за другою схемою ($Q = 1$, операція виділення пам’яті).

| № | Алгоритм (схема, реалізація) | $F, \%$ | $T_{min}$ | $T_{max}$ | $T_{\text{сер}}$ | $S$ | $K, \%$ |
|---|---|---|---|---|---|---|---|
| 1 | Best-Fit (heap2.c) | 64 | 183 | 2555 | 1611 | 527 | 33 |
| 2 | DL malloc (heap3.c) | 36 | 199 | 573 | 287 | 110 | 38 |
| 3 | First-Fit (heap4.c) | 36 | 204 | 332 | 257 | 12 | 5 |
| 4 | First-Fit (heap5.c) | 40 | 198 | 326 | 254 | 20 | 8 |
| 5 | TLSF (C) | 36 | 290 | 803 | 587 | 74 | 13 |
| 6 | TLSF (C + ASM) | 36 | 229 | 374 | 360 | 26 | 7 |

Табл. 3 – Дослідження алгоритмів ДРП за другою схемою ($Q = 20$, операція виділення пам’яті).

| № | Алгоритм (схема, реалізація) | $F, \%$ | $T_{min}$ | $T_{max}$ | $T_{\text{сер}}$ | $S$ | $K, \%$ | $Th$ | $K_d$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Best-Fit (heap2.c) | 38,3 | 183 | 4563 | 2674 | 764 | 29 | 3572 | 121.00 |
| 2 | DL malloc (heap3.c) | 0,0 | 199 | 573 | 294 | 95 | 32 | 509 | 15.75 |
| 3 | First-Fit (heap4.c) | 0,1 | 204 | 394 | 279 | 32 | 11 | 371 | 0.83 |
| 4 | First-Fit (heap5.c) | 0,1 | 198 | 382 | 272 | 31 | 11 | 359 | 0.80 |
| 5 | TLSF (C) | 0,1 | 365 | 818 | 639 | 80 | 13 | 806 | 1.62 |
| 6 | TLSF (C + ASM) | 0,1 | 225 | 373 | 352 | 42 | 12 | 373 | 0.00 |

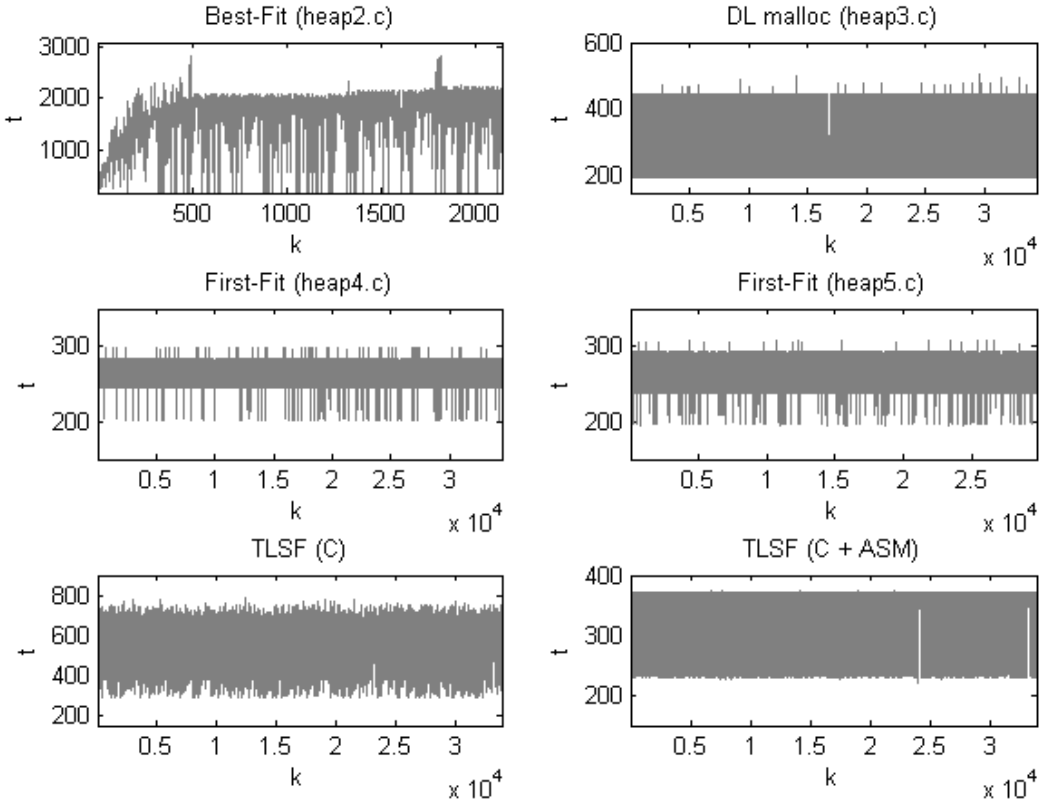


Рис. 2 – Дослідження алгоритмів ДРП за першою схемою ($Q = 1$, операція виділення пам’яті).

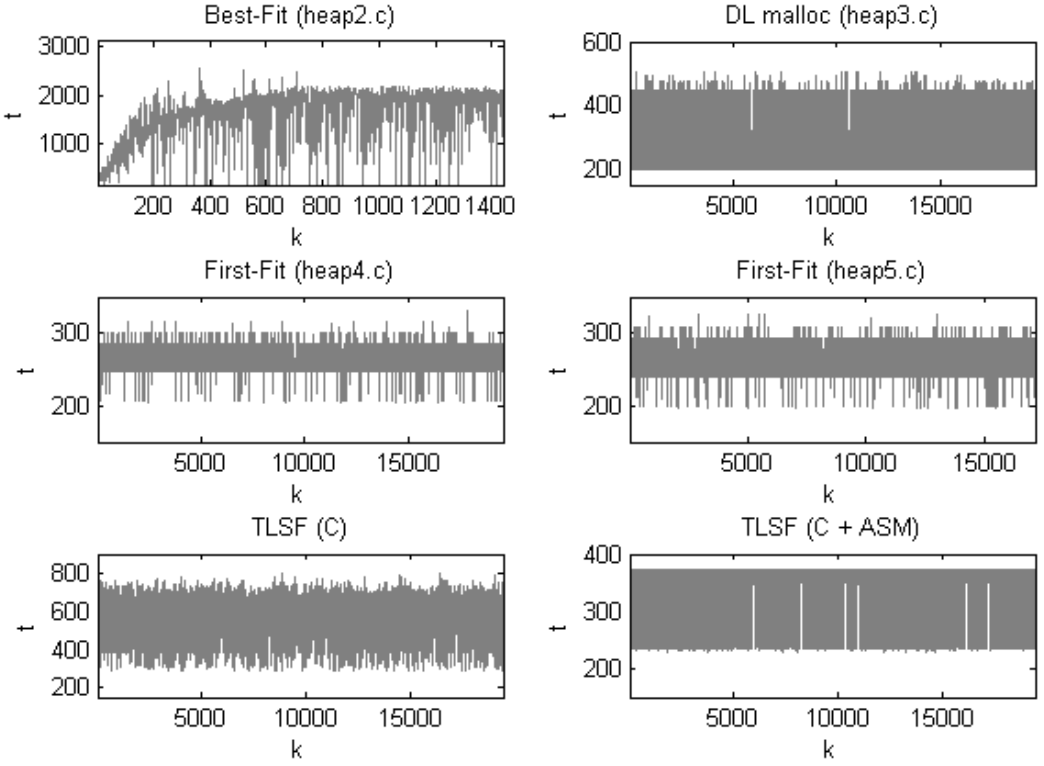


Рис. 3 – Дослідження алгоритмів ДРП за другою схемою ($Q = 1$, операція виділення пам’яті).

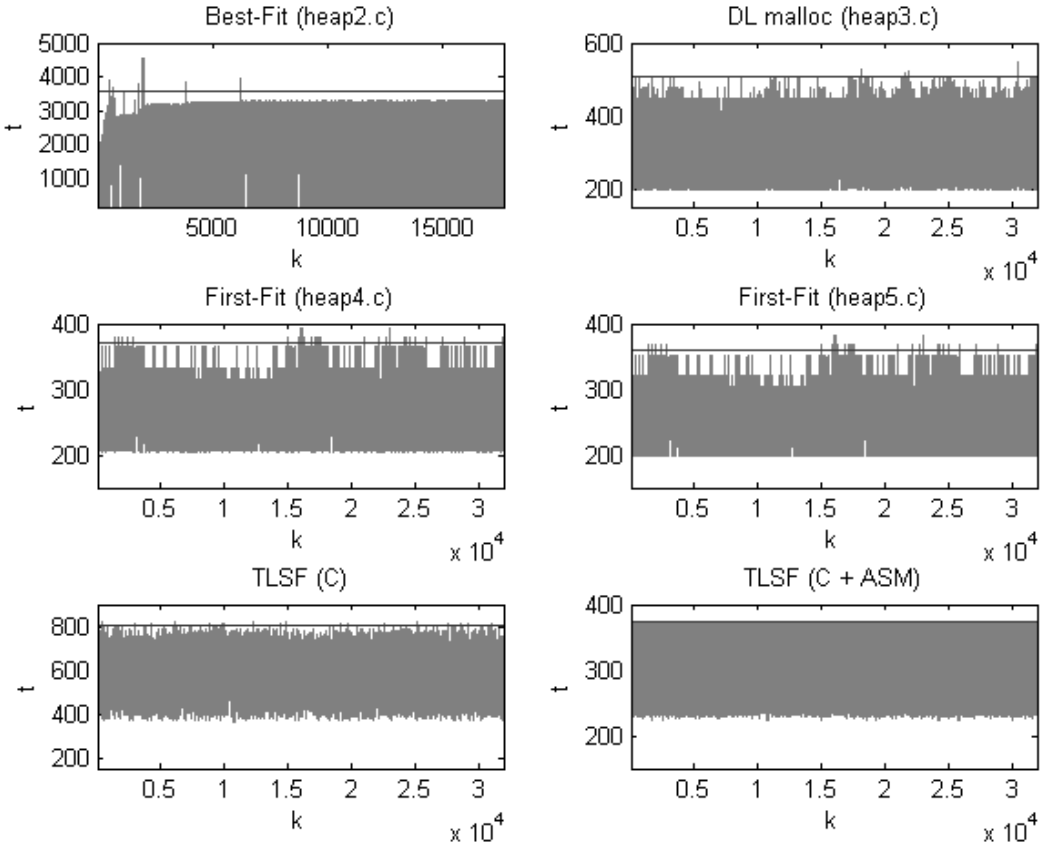


Рис. 4 – Дослідження алгоритмів ДРП за другою схемою ($Q = 20$, операція виділення пам’яті).

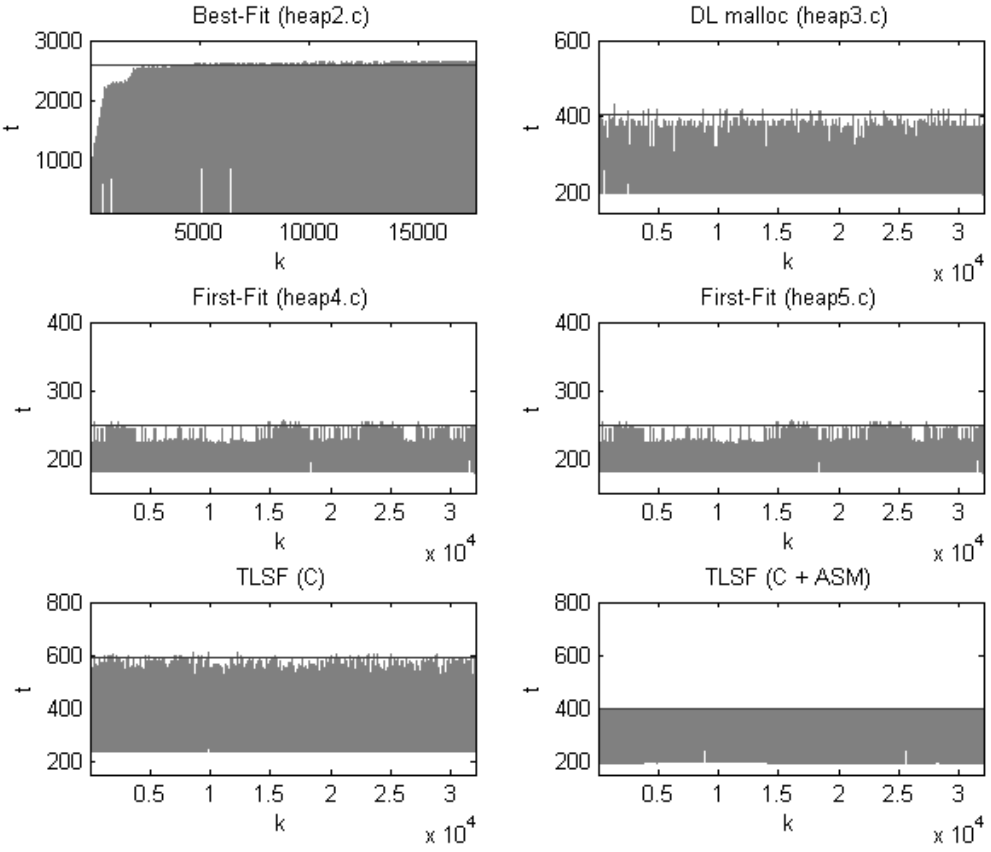


Рис. 5 – Дослідження алгоритмів ДРП за другою схемою ($Q = 20$, операція звільнення пам’яті).

Табл. 4 – Дослідження алгоритмів ДРП за другою схемою ($Q = 20$, операція звільнення пам'яті).

| № | Алгоритм (схема, реалізація) | $T_{min}$ | $T_{max}$ | $T_{\text{сер}}$ | $S$ | $K, \%$ | $Th$ | $K_d$ |
|---|---|---|---|---|---|---|---|---|
| 1 | Best-Fit (heap2.c) | 144 | 2644 | 2135 | 611 | 29 | 2587 | 0.96 |
| 2 | DL malloc (heap3.c) | 196 | 434 | 248 | 53 | 21 | 406 | 2.11 |
| 3 | First-Fit (heap4.c) | 178 | 256 | 211 | 14 | 7 | 249 | 0.17 |
| 4 | First-Fit (heap5.c) | 178 | 256 | 211 | 14 | 7 | 249 | 0.17 |
| 5 | TLSF (C) | 236 | 610 | 369 | 77 | 21 | 593 | 1.28 |
| 6 | TLSF (C + ASM) | 193 | 398 | 283 | 50 | 18 | 397 | 0.00 |

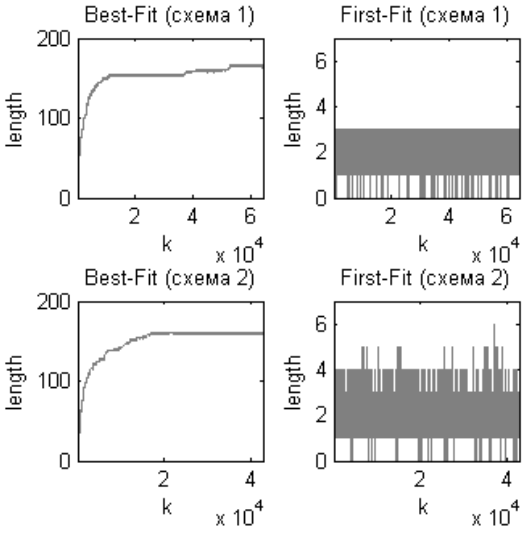


Рис. 6 – Зміна довжини списку вільних блоків в алгоритмах Best-Fit та First-Fit протягом часу моделювання, в залежності від схеми експерименту

*Частотний аналіз алгоритму TLSF* (операція виділення пам'яті, реалізація з асемблерними вставками, схема 1, $Q = 1$). Було виділено три окремих випадки (один випадок відповідає одній гілці або сукупності гілок алгоритму): F1 (відносна частота 0.39) – індекси fi та si [1], розраховані на підставі запитаного розміру блоку, відповідають не порожньому списку вільних блоків; F2 (відносна частота 0.61) – індекси fi та si вказують на порожній список, і їх необхідно корегувати, виконуючи пошук найближчого не порожнього списку з блоками більшого розміру; F3 (відносна частота 0.03) – виділення знайденого вільного блоку без відокремлення надлишкової частини (рис. 7).

Таким чином, на графіку F1 ми бачимо випадки «прямого влучання». При цьому, надлишкова частина може відділятися, або ні. Графік F2 відповідає випадкам промаху. Надлишкова частина може відділятися, або ні (зазвичай, відділяється, але декілька точок внизу відповідають випадкам, коли блок одразу повертається розподільником). F3 – «пряме влучання» або промах, але без відокремлення блоку. Графіки F1-F3 дещо зсунуті з причини появи у коді вставок трасування.

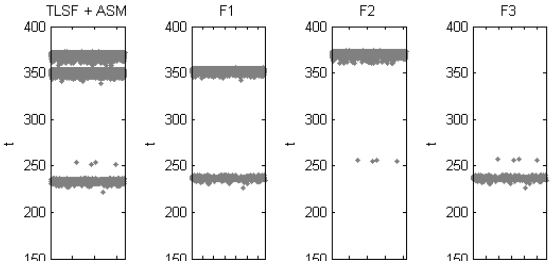


Рис. 7 - Частотний аналіз алгоритму TLSF.

*Частотний аналіз алгоритму First-Fit* (операція виділення пам'яті, схема 1, $Q = 1$). Виділено чотири окремих випадки (рис. 8): F1 (відносна частота 0.997) – випадок з відокремленням надлишкової частини; F2 (відносна частота 0.003) – без відокремлення; F3 (відносна частота 0.167) - з відокремленням надлишкової частини, без об'єднання з наступним вільним блоком; F4 (відносна частота 0.830) - з відокремленням надлишкової частини, з об'єднанням з наступним вільним блоком.

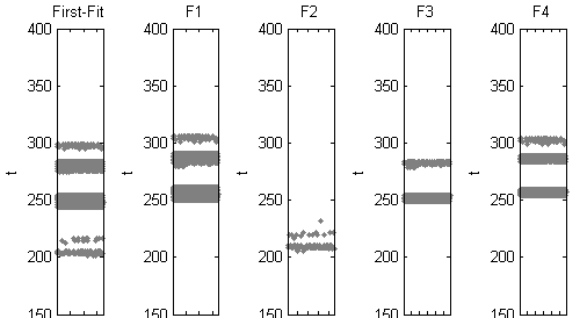


Рис. 8 - Частотний аналіз алгоритму First-Fit.

*Вплив константи SLI [1] на показники алгоритму TLSF*. Константа FLI визначає кількість бітових мап першого рівня (2^FLI), в той час як константа SLI визначає кількість бітових мап другого рівня (2^SLI) для кожного діапазону розмірів блоків. З урахуванням того, що використовувана апаратна платформа містить лише 20 Кб SRAM, було досліджено залежність показників алгоритму лише від значення константи SLI в діапазоні [1; 4] (табл. 5).

Табл. 5 – Вплив константи SLI на показники TLSF (операція виділення пам'яті, схема 1, $Q = 1$).

| SLI | $F, \%$ | $T_{min}$ | $T_{max}$ | $T_{\text{сер}}$ | $S$ | $K, \%$ |
|---|---|---|---|---|---|---|
| 4 | 47 | 222 | 373 | 358 | 25 | 7 |
| 3 | 48 | 225 | 372 | 354 | 32 | 9 |
| 2 | 50 | 222 | 373 | 352 | 36 | 10 |
| 1 | 54 | 224 | 372 | 347 | 42 | 12 |

Бачимо, що зменшення SLI веде до збільшення коефіцієнту коваріації, дещо зменшує середнє значення часу виконання операції, та майже не впливає на $T_{min}$ і $T_{max}$.

*Вплив вирівнювання розміру блоку на показники алгоритмів Best-Fit та First-Fit*. FreeRTOS дозволяє задавати вирівнювання розміру блоку пам'яті, рівне ступеню двійки від 1 до 32. З отриманих результатів (табл. 6) видно, що у разі Best-Fit, вирівнювання дозволяє істотно скоротити час виділення блоку, та дещо скоротити відсоток відмов. У разі First-Fit, вирівнювання дозволяє дещо скоротити час виділення блоку, і практично не впливає на кількість відмов.

*Паралельний виклик функцій розподільника TLSF з основної програми та обробника переривань таймера*. Результати вимірів часу виклику функції malloc() алгоритму TLSF із циклу в основній програмі (схема 2) наведено на рис. 9. Завдяки тому, що основний код функцій malloc() і free() алгоритму TLSF в операційній системі Predicate OS захищений критичними секціями, система працює коректно, хоча видно, що середній час виконання malloc() зріс майже вдвічі, і детермінізм часу її виконання погіршився.

Табл. 6 – Вплив вирівнювання розміру блоку на показники алгоритмів Best-Fit та First-Fit.

| Вирівнювання,байт \ Алгоритм | 1 | | | 2 | | | 4 | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{\text{сер}}$ | $T_{max}$ | $F, \%$ | $T_{\text{сер}}$ | $T_{max}$ | $F, \%$ | $T_{\text{сер}}$ | $T_{max}$ | $F, \%$ |
| Best-Fit | 2477 | 5034 | 97 | 2141 | 3509 | 97 | 1937 | 2935 | 97 |
| First-Fit | 260 | 340 | 46 | 259 | 328 | 46 | 254 | 300 | 46 |

Табл. 6 (продовження)

| Вирівнювання,байт \ Алгоритм | 8 | | | 16 | | | 32 | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{\text{сер}}$ | $T_{max}$ | $F, \%$ | $T_{\text{сер}}$ | $T_{max}$ | $F, \%$ | $T_{\text{сер}}$ | $T_{max}$ | $F, \%$ |
| Best-Fit | 1601 | 2702 | 97 | 1108 | 1742 | 95 | 649 | 1025 | 94 |
| First-Fit | 254 | 300 | 46 | 255 | 301 | 46 | 254 | 301 | 46 |

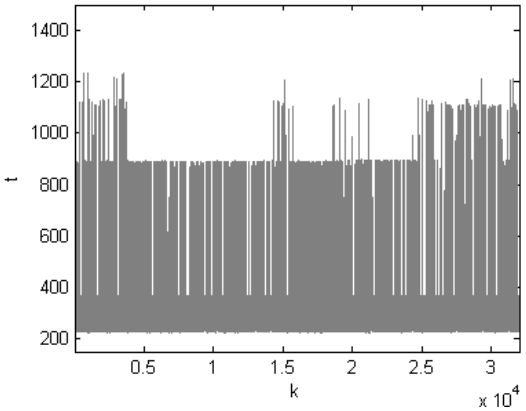


Рис. 9 – Час виклику malloc() TLSF з основної програми, за умови паралельних викликів функцій розподільника із обробника переривань таймера.

*Оцінка часової складності алгоритмів ДРП*. Результати оцінки часової складності розглянутих у роботі алгоритмів ДРП наведено в табл. 7-8 (відповідно, для функцій виділення та звільнення пам'яті).

Табл. 7 – Асимптотична складність за часом функцій виділення пам'яті в різних алгоритмах ДРП

| № | Алгоритм (схема) | Ω | θ | O |
|---|---|---|---|---|
| 1 | Best-Fit (heap2.c) | Ω(1) | θ(n) | O(n) |
| 2 | DL malloc (heap3.c) | Ω(1) | θ(1) | O(1) |
| 3 | First-Fit (heap4.c, heap5.c) | Ω(1) | θ(n) | O(n) |
| 4 | TLSF | Ω(1) | θ(1) | O(1) |

Табл. 8 – Асимптотична складність за часом функцій звільнення пам'яті в різних алгоритмах ДРП

| № | Алгоритм (схема) | Ω | θ | O |
|---|---|---|---|---|
| 1 | Best-Fit (heap2.c) | Ω(1) | θ(n) | O(n) |
| 2 | DL malloc (heap3.c) | Ω(1) | θ(1) | O(1) |
| 3 | First-Fit (heap4.c, heap5.c) | Ω(1) | θ(n) | O(n) |
| 4 | TLSF | Ω(1) | θ(1) | O(1) |

*Накладні витрати пам'яті на зберігання службових даних розподільника* визначаються в байтах відповідно до наведених нижче виразів (в залежності від схеми / алгоритму динамічного розподілу пам'яті), в яких $n$ – кількість блоків, на які фрагментовано купу.

Best-Fit (heap2.c): $M = 22 + 8n$.

First-Fit (heap4.c): $M = 24 + 8n$.

First-Fit (heap5.c): $M = 24 + 8(k + 1) + 8n$, де $k$ – кількість регіонів пам'яті.

TLSF: $M = 20 + 4 \cdot 2^{FLI} \cdot (1 + 2^{SLI}) + 16n$, або $M = 1108 + 16n$, при $FLI = 4$ та $SLI = 4$.

DL malloc: $M = 2196 + 16n_1 + 32n_2$, де $n_1$ – кількість блоків розміром менше ніж 256 байт (smallbins) [6, 11, 17], $n_2$ – кількість блоків розміром 256 байт або більше (largebins).

З метою порівняння, в табл. 9 наведено значення накладних витрат пам'яті $M$ для різних схем / алгоритмів ДРП, виходячи з припущення, що в системі наявно 10 блоків розміром 200 байт, та 10 блоків розміром 300 байт. Позначення в таблиці: $M_1, \%$ - накладні витрати пам'яті у відсотках від загального об'єму пам'яті для мікроконтролеру з 20 Кб SRAM; $M_2, \%$ - те ж саме для плати Raspberry Pi 3 з 1 Гб RAM (значення $M_1$ і $M_2$ округлені та наближені).

*Списки*. Функції для роботи зі списками в Predicate OS мають константний час. На рис. 10 (а, б) показано час додання нового елементу до списку для двох випадків: (а) – список було створено з обмеженням на кількість елементів, які він може містити (n=20; якщо список містить максимально допустиму кількість елементів, при спробі додання ще одного, функція одразу повертає управління), (б) – без обмеження. За аналогією, на рис. 10 (г, ґ), наведено результати для вилучення елементу зі списку. Функції для роботи зі списками в FreeRTOS також мають константний час, за виключенням функції додання елементу до списку vListInsert(), найгірший час виконання якої становить O(n) (рис. 10, в). На рис. 10 (д) показано час видалення елементу зі списку в FreeRTOS. Часові показники функцій роботи зі списками в Predicate OS та FreeRTOS наведено в табл. 10.

Табл. 9 – Накладні витрати памяті у разі використанні різних алгоритмів ДРП.

| Алгоритм (схема) | $M$ | $M_1, \%$ | $M_2, \%$ |
|---|---|---|---|
| Best-Fit (heap2.c) | 182 | 1 | $10^{-5}$ |
| First-Fit (heap4.c) | 180 | 1 | $10^{-5}$ |
| First-Fit (heap5.c), $(k = 2)$ | 200 | 1 | $10^{-5}$ |
| TLSF | 1428 | 7 | $10^{-4}$ |
| DL malloc | 2676 | 13 | $10^{-4}$ |

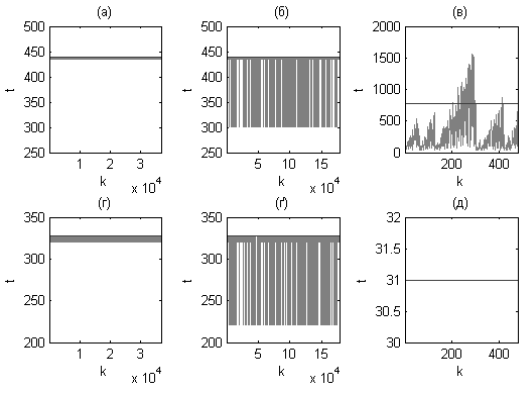


Рис. 10 – Час виконання функцій роботи зі списками в Predicate OS та FreeRTOS.

Табл. 10 – Часові показники функцій роботи зі списками в Predicate OS та FreeRTOS.

| № | ОС | Операція | Функція | $T_{min}$ | $T_{max}$ | $T_{\text{сер}}$ | $Th$ | $K_d$ |
|---|---|---|---|---|---|---|---|---|
| 1 | Predicate OS | add | add_list_item (з обмеженням) | 435 | 439 | 439 | 439 | 0 |
| 2 | | add | add_list_item | 302 | 439 | 439 | 439 | 0 |
| 3 | | delete | delete_list_item (з обмеженням) | 320 | 328 | 328 | 328 | 0 |
| 4 | | delete | delete_list_item | 221 | 328 | 328 | 328 | 0 |
| 5 | FreeRTOS | add | vListInsert | 45 | 1563 | 297 | 774 | 63 |
| 6 | | delete | uxListRemove | 31 | 31 | 31 | 31 | 0 |

Результати оцінки накладних витрат пам'яті $M$ в байтах на службові дані для роботи зі списками в обох досліджуваних ОС наведено в табл. 11. Позначення, використані в таблиці: $n$ – кількість елементів в списку, $M,\%$ - відношення $M$ до загального об'єму оперативної пам'яті, для випадку $n = 20$ та об'єму SRAM 20 Кб.

Табл. 11 – Накладні витрати пам'яті при роботі зі списками.

| ОС | $M$ | $M, n = 20$ | $M,\%$ |
|---|---|---|---|
| Predicate OS | $16 + 36n$ | 736 | 4 |
| FreeRTOS | $20 + 28n$ | 580 | 3 |

*Черги*. Всі функції для роботи з чергами в обох досліджуваних ОС мають константний час. Час виконання операцій додання (enqueue) та видалення (dequeue) елементу з черги в FreeRTOS наведено на рис. 11 (в) і (д) відповідно. В Predicate OS можливе створення черг з пріоритетами або без. Час виконання операцій додання та видалення елементу з черги для випадків черги без пріоритетів та з пріоритетами наведено на рис. 11 (а, б) та (г, ґ) відповідно. Часові показники функцій роботи з чергами в Predicate OS та FreeRTOS наведено в табл. 12.

Накладні витрати при роботі з чергами в FreeRTOS залежать від конфігурації проекту. У разі мінімальної конфігурації (не використовується групування черг, вимкнені функції налагодження тощо), вони становлять 72 байти на одну чергу.

Накладні витрати при роботі з чергами в Predicate OS визначаються виразом (5):

$$M = 16 \cdot mqc + 17 + 32 \cdot qc + \quad (5)$$
$$+ 36 \cdot (sn + pq),$$

де $mqc$ – максимально можлива кількість черг в системі, визначена у файлі конфігурації, $qc$ – загальна кількість черг в системі, $sn$ – сумарна кількість елементів в всіх чергах, $pq$ – кількість черг з пріоритетами. Так, у разі наявності в системі однієї черги без пріоритетів, що містить 10 елементів, при $mqc = 32$ отримаємо 597 байт.

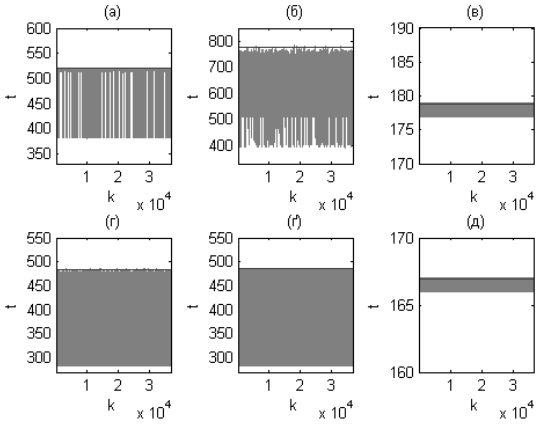


Рис. 11 – Час виконання функцій роботи з чергами в Predicate OS та FreeRTOS.

Табл. 12 – Часові показники функцій роботи з чергами в Predicate OS та FreeRTOS.

| № | ОС | Операція | $T_{min}$ | $T_{max}$ | $T_{\text{сер}}$ | $Th$ | $K_d$ |
|---|---|---|---|---|---|---|---|
| 1 | Predicate OS | enqueue | 380 | 520 | 518 | 520 | 0,00 |
| 2 | | enqueue (з пріорітетами) | 392 | 784 | 602 | 777 | 1,36 |
| 3 | | dequeue | 283 | 485 | 385 | 484 | 0,26 |
| 4 | | dequeue (з пріорітетами) | 283 | 486 | 376 | 486 | 0,00 |
| 5 | FreeRTOS | enqueue | 177 | 179 | 177 | 179 | 0,004 |
| 6 | | dequeue | 166 | 166 | 166 | 166 | 0,002 |

**Висновки**. Було виконано дослідження та порівняльний аналіз алгоритмів ДРП та базових структур даних, що використовуються в операційних системах FreeRTOS і Predicate OS. На підставі отриманих результатів, можна зробити наступні висновки:

1. Якщо один алгоритм характеризується часом O(1), а другий часом O(n), то це ще не означає, що на практиці детермінізм першого буде кращим. Наприклад, DL malloc має найгірший час виконання O(1), а First-Fit - O(n). Проте, коефіцієнт детермінованості першого дорівнює 15.75, а для First-Fit – 0.83 (табл. 3).

2. Якщо алгоритм характеризується часом O(n), це ще не означає, що на практиці він має поганий детермінізм. Наприклад, в реальних умовах, довжина списку вільних блоків в алгоритмі First-Fit змінюється в

досить вузькому діапазоні (рис. 6), що робить його майже детермінованим.

3. Використання асемблерних вставок із застосуванням команд CLZ та RBIT із системи команд Cortex-M3 дозволяє досягти практично ідеального детермінізму в разі алгоритму TLSF.

4. Зі збільшенням коефіцієнту SLI в алгоритмі TLSF розкид часу виділення блоку зменшується.

5. У разі використання алгоритму Best-Fit, вирівнювання розміру блоку на 32 байти дозволяє суттєво скоротити час виділення пам'яті. Однак, з огляду на великий час виконання алгоритму Best-Fit та сильну не детермінованість, його використання вбачається нераціональним (тим паче, в задачах, що потребують роботи в режимі реального часу).

6. Небажано використовувати функції динамічного управління пам'яттю всередині обробників переривань, оскільки це може суттєво погіршити реактивні властивості та детермінованість системи.

7. Використання списків та черг в Predicate OS пов'язане із значно більшими накладними витратами пам'яті, ніж у FreeRTOS. Питання про те, чи виправдані такі витрати на практиці, потребує додаткового дослідження.

Варто також відзначити різницю між алгоритмом та його імплементацією, і пам'ятати про неї при дослідженні часових властивостей алгоритмів, адже час виконання може істотно залежати не тільки від логіки алгоритму, але й від способу його реалізації.

Отримані в даній роботі результати можуть допомогти практичним програмістам більш свідомо підходити до вибору алгоритмів ДРП для своїх проектів та їх параметрів. Запропоновані поняття умовної верхньої межі часу виконання операції та коефіцієнт детермінованості мають теоретичну та практичну цінність, і дозволяють кількісно оцінювати детермінізм імплементованих алгоритмів.

## Література


[1] M. Masmano, I. Ripoll, A. Crespo, J. Real, "TLSF : новая система распределения динамической памяти для систем реального времени."

[2] R. Barry, *Using the FreeRTOS Real Time Kernel*. 2009.

[3] К. С. Гайдук, О. Г. Шевченко, "Предотвращение взаимоблокировок в ОС для программируемых систем на кристалле," *Sci. Educ. a New Dimens. Nat. Tech. Sci.*, т. 6, № 172, С. 25–28, 2018, doi: 10.31174/SEND-NT2018-172VI20-06.

[4] K. S. Gaiduk, O. G. Shevchenko, "The Deadlock Problem & Approaches to Its Solution," in *Computer and information systems and technologies*, 2018, С. 59.

[5] "FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions," 2019. [Online]. Режим доступу: https://www.freertos.org/. [Перевірено: 25.12.2019].

[6] J. Hemanth, X. Fernando, P. Lafata, and Z. Baig, "International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018," 2019, т. 26, С. 1053–1060, doi: 10.1007/978-3-030-03146-6.

[7] V. Heikkilä, "A Study on Dynamic Memory Allocation Mechanisms for Small Block Sizes in Real-Time Embedded Systems," University of Oulu, 2012.

[8] M. Masmano, I. Ripoll, and A. Crespo, "Dynamic storage allocation for real-time embedded systems," *Real-time Syst. Symp. Work. Sess.*, С. 1–4, 2003.

[9] S. S. Craciunas, A. Sokolova, C. M. Kirsch, H. Stadler, H. Payer, and R. Staudinger, "A Compacting Real-Time Memory Management System," *Proc. 2008 USENIX Annu. Tech. Conf. USENIX 2008*, С. 349–362, 2008.

[10] W. Fang, "Analysis on Dynamic Memory Allocation," С. 1–17, 2012.

[11] A. Abraham, P. Dutta, J. K. Mandal, A. Bhattacharya, and S. Dutta, "Emerging Technologies in Data Mining and Information Security," in *Proceedings of IEMIS 2018*, 2018, С. 889.

[12] Э. Таненбаум, *Современные операционные системы*, 3rd ed. СПб.: Питер, 2010.

[13] "FreeRTOS - Memory management options for the FreeRTOS small footprint, professional grade, real time kernel (scheduler)," 2019. [Online]. Режим доступу:

https://www.freertos.org/a00111.html. [Перевірено: 25.12.2019].
[14] К. С. Гайдук, "Схеми дослідів," 2019. [Online]. Режим доступу: https://github.com/ks-gayduk/dma_test. [Перевірено: 25.12.2019].
[15] К. С. Гайдук and О. Г. Шевченко, "Аналітичний огляд парадигм подійно-орієнтованого та автоматного програмування," in *Інформаційна безпека та комп'ютерні технології: Збірник тез доповідей III Міжнародної науково-практичної конференції, 19-20 квітня 2018 року*, 2018, С. 336.
[16] "FreeRTOS API categories," 2019. [Online]. Режим доступу: https://www.freertos.org/a00106.html. [Перевірено: 25.12.2019].
[17] D. Lea, "dlmalloc." [Online]. Режим доступу: http://gee.cs.oswego.edu/pub/misc/malloc-2.8.4.c. [Перевірено: 25.12.2019].

**References**

[1] M. Masmano, I. Ripoll, A. Crespo, and J. Real, " TLSF: A new dynamic memory allocator for real-time systems."
[2] R. Barry, *Using the FreeRTOS Real Time Kernel*. 2009.
[3] K. S. Gaiduk, O. G. Shevchenko, " Deadlock Prevention in OS for Programmable Systems on a Chip," *Sci. Educ. a New Dimens. Nat. Tech. Sci.*, vol. 6, no. 172, pp. 25–28, 2018, doi: 10.31174/SEND-NT2018-172VI20-06.
[4] K. S. Gaiduk, O. G. Shevchenko, "The Deadlock Problem & Approaches to Its Solution," in *Computer and information systems and technologies*, 2018, p. 59.
[5] "FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions," 2019. [Online]. Available: https://www.freertos.org/. [Accessed: 25-Dec-2019].
[6] J. Hemanth, X. Fernando, P. Lafata, and Z. Baig, "International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018," 2019, vol. 26, pp. 1053–1060, doi: 10.1007/978-3-030-03146-6.
[7] V. Heikkilä, "A Study on Dynamic Memory Allocation Mechanisms for Small Block Sizes in Real-Time Embedded Systems," University of Oulu, 2012.
[8] M. Masmano, I. Ripoll, and A. Crespo, "Dynamic storage allocation for real-time embedded systems," *Real-time Syst. Symp. Work. Sess.*, pp. 1–4, 2003.
[9] S. S. Craciunas, A. Sokolova, C. M. Kirsch, H. Stadler, H. Payer, and R. Staudinger, "A Compacting Real-Time Memory Management System," *Proc. 2008 USENIX Annu. Tech. Conf. USENIX 2008*, pp. 349–362, 2008.
[10] W. Fang, "Analysis on Dynamic Memory Allocation," pp. 1–17, 2012.
[11] A. Abraham, P. Dutta, J. K. Mandal, A. Bhattacharya, and S. Dutta, "Emerging Technologies in Data Mining and Information Security," in *Proceedings of IEMIS 2018*, 2018, p. 889.
[12] Andrew S. Tanenbaum, *Modern Operating Systems*, 3rd ed. SPb.: Piter, 2010.
[13] "FreeRTOS - Memory management options for the FreeRTOS small footprint, professional grade, real time kernel (scheduler)," 2019. [Online]. Available: https://www.freertos.org/a00111.html. [Accessed: 25-Dec-2019].
[14] K. S. Gaiduk, " Schemes of experiments," 2019. [Online]. Available: https://github.com/ks-gayduk/dma_test. [Accessed: 25-Dec-2019].
[15] K. S. Gaiduk, O. G. Shevchenko, An analytical overview of event-oriented and automata-based programming paradigms. Information Security and Computer Technologies: a collection of abstracts of the 3nd International Science and Practical Conference (Kropivnitsky, April 19-20, 2018), p. 336.
[16] "FreeRTOS API categories," 2019. [Online]. Available: https://www.freertos.org/a00106.html. [Accessed: 25-Dec-2019].
[17] D. Lea, "dlmalloc." [Online]. Available: http://gee.cs.oswego.edu/pub/misc/malloc-2.8.4.c. [Accessed: 25-Dec-2019].

*Надійшла до редакції 25.12.2019 р.*

**Гайдук Кирило Сергійович** – аспірант, Донецький національний технічний університет (пл. Шибанкова, 2, м. Покровськ, 85300, Україна).
E-mail: kyrylo.haiduk@donntu.edu.ua.
**Шевченко Ольга Георгіївна** – старший викладач, Донецький національний технічний університет (пл. Шибанкова, 2, м. Покровськ, 85300, Україна).
E-mail: olha.shevchenko@donntu.edu.ua.

## DATA STRUCTURES AND MEMORY MANAGEMENT IN THE FREERTOS AND PREDICATE OS OPERATING SYSTEMS

*The results of the study and comparative analysis of dynamic memory allocation algorithms (DMA) and basic data structures used in the FreeRTOS and Predicate OS embedded operating systems are presented. These systems use such algorithms of DMA as Best-Fit, First-Fit, DL malloc and TLSF. Dynamic lists and queues are the basic data structures of these operating systems. The following indicators were used to quantify and compare algorithms of DMA: minimum, average, and maximum values of operation time; standard deviation based on offset variance estimate; covariance coefficient. Although these indicators provide some characteristic of the algorithm, they are uninformative in the context of deterministic estimation, since in the latter case, the largest variation is the variation of the random variable relative to some upper bound rather than the average. In this connection, the notion of conditional upper limit of operation time, as well as the coefficient of determination, is introduced in the paper and an algorithm for their calculation is given. Two variants of TLSF algorithm implementation are considered: exclusively in C language and using assembly insertions. It is shown that the use of CLZ and RBIT commands from the Cortex-M3 command system allows to achieve practically perfect determinism of the algorithm operation. Frequency analysis and analysis of the influence of the parameters (constants) of the algorithms on their time and statistical performance were performed for individual algorithms of DMA. Additionally, algorithms of DMA and lists and queue functions have been evaluated for their asymptotic time complexity and memory overhead for storing service data. The studies were performed using the STM32F103C8 (20 Kb SRAM) platform, using the high resolution DWT timer. The results obtained can be useful for software engineers in selecting the algorithms for dynamic memory allocation and their parameters.*

***Keywords:*** *dynamic memory allocation algorithms, determinism of algorithm, dynamic lists, queues, resource overhead, real-time operating systems.*

**Haiduk Kyrylo Serhiiovych** – PhD-student, Donetsk National Technical University, (2, Shybankova square, Pokrovsk, Donetsk region, 85300, Ukraine).
E-mail: kyrylo.haiduk@donntu.edu.ua.
**Shevchenko Olha Heorhiivna** – Senior Lecturer, Donetsk National Technical University, (2, Shybankova square, Pokrovsk, Donetsk region, 85300, Ukraine).
E-mail: olha.shevchenko@donntu.edu.ua.