УДК 004.432.2

ДОСЛІДЖЕННЯ МОЖЛИВОСТЕЙ ОПЕРАЦІЙНОЇ СИСТЕМИ MONGOOSE OS НА ПЛАТФОРМІ ESP8266

Мельник Дмитро

Науковий керівник: канд. техн. наук, доцент Баранюк О.Ф.

Центральноукраїнський державний педагогічний університет

імені Володимира Винниченка, м. Кропивницький, Україна

У статті проаналізовано можливості вбудованого програмного забезпечення Інтернету речей Mongoose OS, доступної під ліцензією Apache версії 2.0. Проведено порівняння кількісної характеристики такої як швидкість виконання програм та якісних характеристик, зокрема легкість написання та простота читання коду написаних мовами JavaScript з двигуном mJS та C. Також описано процедуру прискорення роботи JavaScript за рахунок винесення складних математичних обчислень в функції написані на C та використання їх за допомогою експорту користувацьких функції (FFI).

Ключові слова: Mongoose OS, ESP8266, JavaScript, C, вбудовані системи.

RESEARCH OF MONGOOSE OS CAPABILITIES ON ESP8266 PLATFORM

D. Melnik

Scientific supervisor: Candidate of Technical Sciences, Associate Professor

Baranyuk O. F.

Volodymyr Vynnychenko Central Ukrainian State Pedagogical University,

Kropyvnytskyi, Ukraine

The article analyzes the capabilities of the built-in software Mongoose OS, available under the Apache version 2.0 license. Also quantitative characteristics such as speed of execution and qualitative characteristics, in particular, ease of writing and simplicity of reading code written in JavaScript with the engine mJS and C was compared. It was later found that we can accelerate the work of JavaScript due to the removal of complex mathematical calculations in the functions written in C and use them by exporting custom functions (FFI).

Keywords: Mongoose OS, ESP8266, JavaScript, C, embedded systems.

Постановка проблеми. На сучасний момент найактуальніші мови програмування для вбудованих систем це асемблер, C та C++. Вони дозволяють найбільш ефективно керувати апаратними засобами мікроконтролерів.

Асемблер — мова програмування низького рівня, що являє собою формат запису машинних команд, зручний для сприйняття людиною.

Асемблер має головну перевагу над іншими мовами — це швидкість виконання програм, так як програма це набір машинних команд написаних спеціально під конкретну архітектуру. Але ця вузькоспеціалізованість є і основним його недоліком — відсутність абстракції робить розробку досить тривалим процесом.

C — універсальна, процедурна, імперативна мова програмування загального призначення.

C++ — мова програмування високого рівня з підтримкою кількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної.

Програми написані на C/C++ зазвичай вже більш універсальні та мають деякий рівень абстракції.

Але при розробці прототипу доводиться використовувати одну з подібних мов, що є не зовсім доцільним, так як основною задачею створення прототипу є зробити це в якомога коротші часові рамки з найбільшою кількістю функціоналу.

У свою чергу операційні системи теж додають рівень абстракції, що дозволяє розробникам не займатись кожен раз рутинними проблемами, а займатись лише розробкою функціональної частини програми.

Mongoose OS дозволяє розробляти додатки ефективніше, приховавши від розробника такі речі як контроль пам’яті, підтримку різних платформ (Mongoose OS підтримує STM32 L3/F4/F7, TI CC 3220/3200, ESP 32/8266 та ін.), вбудовану підтримка протоколів (TCP, UDP, HTTP, MQTT, WebSocket, CoAP), ще одним її плюсом є те, що вона дозволяє використовувати мову JavaScript з деякими незначними обмеженнями, а також комбінувати мови JavaScript та C, що дає

розробникам, і простоту, і швидкість розробки JavaScript'а та потужність та швидкість виконання C.

Аналіз досліджень і публікацій.

Тема використання мови C в програмуванні для вбудованих систем розкрита досить добре, тому що це одна з перших мов, яка для них використовувалась, але її порівняння з менш популярними мовами в цій галузі майже не розкрите через те, що вони не так давно почали використовуватись, та все ще є недостатньо популярними. Так, перша версія двигуна Mjs, який виконує JavaScript код, з'явилась лише в кінці 2017 року, проте відтоді зазнала багато змін, відбулось 58 випусків версій, це означає, що робота над ним ведеться досить активно.

Публікацій, присвячених порівняням з іншими мовами не так багато, а офіційна документація Mongoose OS [1] рекомендує використовувати JavaScript лише для розробки прототипів та для ознайомлення з її можливостями, а для розробки продукту використовувати C.

Також є деякі статті та форуми, які не є науковими публікаціями, де підіймаються схожі питання про доцільність використовування альтернативних мов для розробки вбудованих систем.

Lua, Micropython, JavaScript, та інші мови сценаріїв мають тенденцію підвищувати продуктивність праці програміста: все стає набагато простіше, якщо працювати на більш високому рівні абстракції і не турбуватись про такі прості речі, як управління пам'яттю. Можна досягти тих же цілей з набагато меншою кількістю рядків коду. З іншого боку, хоча це й прискорює розробку, але також уповільнює виконання програми. В статті [4] було проведено порівняння Lua, Micropython та Arduino IDE(C++) на мікроконтролері ESP8266, але лише з огляду на частоту, на яку вони спроможні, та прийшли до висновків, що Arduino IDE(C++) показує найкращі результати. Але не висвітленні питання про використання JavaScript і простоту та швидкість написання коду.

Метою статті є аналіз гнучкості та швидкості розробки та швидкості виконання коду написаного мовами C, JavaScript та дослідження можливостей підвищення цих характеристик за допомогою експорту користувацьких функцій (FFI), з використанням усіх переваг обох мов.

Виклад основного матеріалу дослідження.

Порівняння швидкості розробки програм та швидкості їх виконання у даному дослідженні здійснюється шляхом реалізації трьох математичних функції з використанням можливостей JavaScript та C в Mongoose OS.

Нами реалізовані функції для розрахунків числа Пі, факторіалу та n-го числа Фібоначчі. Кожна з функцій написана двома мовами C та JavaScript. Вони будуть викликатись за допомогою RPC (виклик віддалених процедур) це дає нам змогу викликати функції, які були описані в коді та передавати в них параметри.

Наприклад, за допомогою JavaScript такі функції оголошуються так:

```
load('api_rpc.js');
load('api_sys.js');

RPC.addHandler('JSgetPI', function (args) {
  let startTime = Sys.uptime();
  let value = getPI(args.c);
  let endTime = Sys.uptime();
  return {
    value: value,
    time: endTime - startTime
  };
});
```

В даному прикладі необхідно підключити бібліотеку “api_rpc.js”, яка містить об’єкт RPC, та за допомогою функції addHandler вказати назву за якою можна буде звертатись до цієї процедури та саму функцію що буде виконуватись. Функція приймає один аргумент args, в ньому знаходяться всі параметри, які були передані в процедуру. Процедура повинна повернути об’єкт що повернеться в місце виклику процедури у форматі JSON.

За допомогою мови C така процедура оголошується трохи складніше:

```
#include "mgos.h"
#include "mgos_rpc.h"
#include "mgos_time.h"

static void rpc_pi(struct mg_rpc_request_info *ri, void *cb_arg, struct
mg_rpc_frame_info *fi, struct mg_str args) {
  int count = 0;
  json_scanf(args.p, args.len, ri->args_fmt, &count);

  double startTime = mgos_uptime();

  double pi = calculatePI(count);
  double endTime = mgos_uptime();
  mg_rpc_send_responsef(ri, "{value: %f, time: %f}", pi, endTime-startTime);
  (void) fi;
  (void) cb_arg;
}

enum mgos_app_init_result mgos_app_init(void) {
  mg_rpc_add_handler(mgos_rpc_get_global(), "CgetPI", "{c: %d}", rpc_pi, NULL);
  return MGOS_APP_INIT_SUCCESS;
}
```

Спочатку підключаються бібліотеки: “mgos.h” з основними функціями Mongoose OS, “mgos_rpc.h” з функціями для керування RPC та “mgos_time.h” для вимірювання часу виконання віддаленої процедури. Далі йде оголошення функції яка стане віддаленою процедурою, вона приймає параметри що були передані при створенні процедури та при її виклику. В тілі процедури за допомогою функції json_scanf дістаються параметри, що були їй передані. Після виконання тіла процедури потрібно повернути відповідь за допомогою mg_rpc_send_responsef, для цього потрібно вказати формат в якому вона буде повернута та змінні які будуть передані. Також ще однією особливістю є те, що кожна оголошена змінна або функцію повинна бути використана, інакше скомпілювати код не вийде, тому змінні які були передані в функцію, але не використовувались мають такий запис (void) cb_arg. Але це ще не все, процедура описана, але ще потрібно її оголосити як RPC. Для цього в головній функції (зазвичай це main, але в Mongoose OS це

mgos_app_init) потрібно викликати функцію mg_rpc_add_handler яка оголосить процедуру віддаленою, задасть їй ім'я для виклику та задасть правило для вхідних параметрів.

Програма на JavaScript, містить всього 11 рядків коду, в той час як С-програма 18. Отже, за простотою написання й читанням коду перемагає JavaScript.

Крім JavaScript та C, буде ще їх поєднання за допомогою FFI (foreign function interface), це досить проста та ефективна можливість експортувати функції з C в JavaScript.

Перша програма для порівняння швидкості це розрахунок числа Pi:

- Код мовою JavaScript:

```
function calculatePI(count) {
   let i = 1,
      p = 0;
   while (i < count) {
      p += 1 / i - 1 / (i + 2);
      i += 4;
   }
   return 4 * p;
}
```

- Код мовою C:

```
double calculatePI(int count) {
  double i = 1;
  double p = 0;
  while (i < count) {
    p += 1 / i - 1 / (i + 2);
    i += 4;
  }
  return p * 4;
}
```

- Код з використанням FFI:

```
let ffiCalculatePI = ffi('double calculatePI(int)');
```

Таблиця 1. Час виконання обчислень числа Пі

Точність обчислень	Час виконання в секундах			Число Пі
	C	FFI	JS	

50	0.000182	0.003996	0.046571	3.103145
100	0.000326	0.00408	0.082464	3.121595
150	0.000482	0.004238	0.121353	3.128435
200	0.000632	0.004382	0.15725	3.131593
250	0.000789	0.004539	0.196142	3.133656
300	0.000931	0.004684	0.230693	3.134926
350	0.001089	0.004841	0.270932	3.135911
400	0.001234	0.004987	0.306825	3.136593
450	0.00139	0.005144	0.345718	3.137168
500	0.001535	0.005289	0.381614	3.137593
550	0.001696	0.00545	0.420507	3.137969
600	0.001842	0.005594	0.456405	3.138259
650	0.001999	0.005753	0.495298	3.138525
700	0.002144	0.005898	0.531196	3.138736
750	0.002303	0.006057	0.570099	3.138933
800	0.002449	0.006201	0.605991	3.139093
850	0.002605	0.006361	0.644877	3.139245
900	0.00275	0.006504	0.680775	3.13937
950	0.002908	0.006662	0.719666	3.139492
1000	0.003052	0.006861	0.755539	3.139593

З таблиці результатів одразу можна помітити як погано впорався JavaScript з цим завданням.

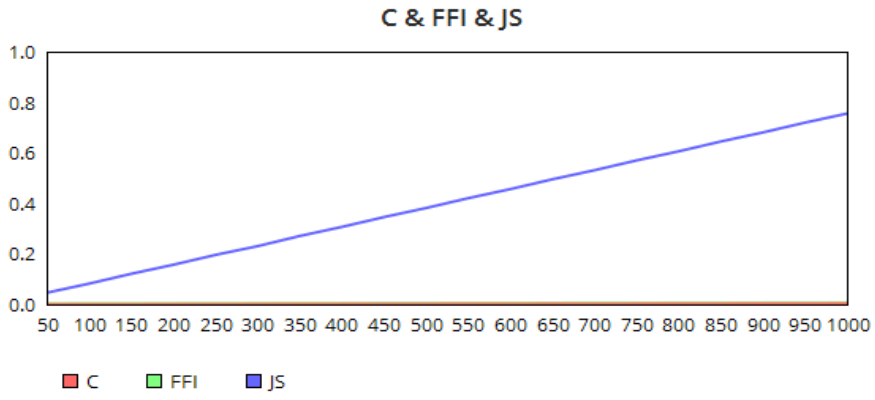

Рис. 1. Витрачений час на розрахунок з точністю n для C, JavaScritpt та FFI (час виконання JavaScript суттєво перевищує час C та FFI).

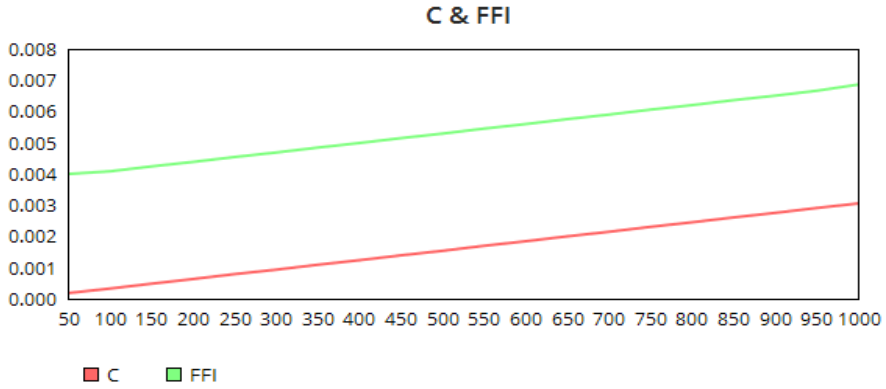

Рис. 2. Витрачений час на розрахунок з точністю n для C та FFI.

Але якщо подивитись на графік то можна помітити, що середній час відставання FFI від C 0.003759, імовірно стільки часу затрачує ESP8266 для виклику функції за допомогою FFI.

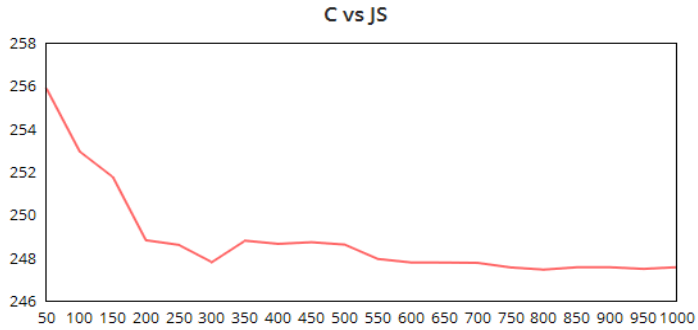

Рис. 3. Відношення швидкості виконання коду C до JavaScript на кожному з варіантів обчислення.

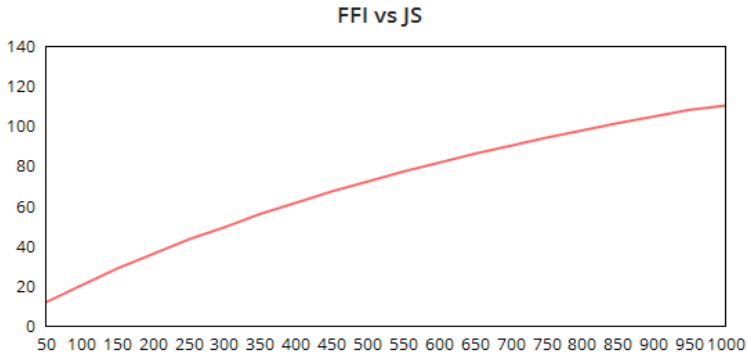

Рис. 4 Відношення швидкості виконання коду з FFI до JavaScript на кожному з варіантів обчислення.

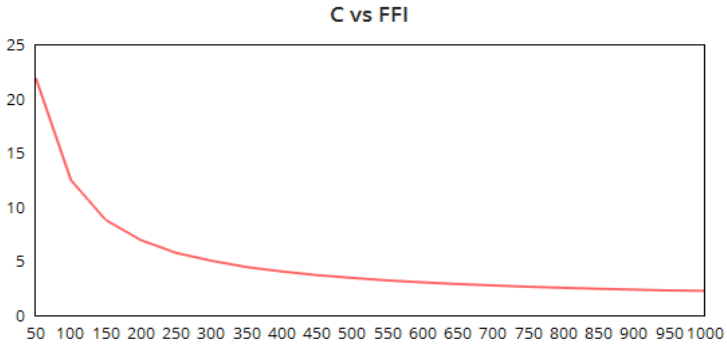

Рис. 5. Порівняння швидкості коду C до швидкості FFI на кожному з варіантів обчислення.

За допомогою графіків, можна прослідкувати ще декілька цікавих залежностей:

- C в середньому виконується у 248 раз швидше ніж JavaScript, при збільшенні точності підрахунків це число падає, але незначно.
- FFI в середньому виконується в 68 раз швидше ніж JavaScript, але при збільшенні кількості підрахунків це число досить швидко росте. Останній етап був підрахований в 110 разів швидше, ніж за допомогою JavaSctipt.
- На першому етапі код на C виконався майже у 22 рази швидше ніж за допомогою FFI, проте на останньому етапі лише у 2 раз. Отже, час на який відстає FFI – це час його виклику, а отже при збільшенні розрахунків час виконання буде наближатися до C.

Друга програма розраховує факторіал.

Код досить простий і працює за однаковим принципом як і на JavaScript так і C.

- JavaScript:

```
function factorial(n) {
   return (n < 2) ? 1 : n * factorial(n - 1);
}
```

- C:

```
uint64_t factorial (int n){
  return (n < 2) ? 1 : n * factorial (n - 1);
}
```

В C для підрахування факторіалу більшого за 12, потрібно вказати, що функція повертає uint64_t (64-біт без знака).

Через великі значення код для FFI трохи ускладнюється через те, що він може приймати лише прості значення (int, double, char *, void *), тому потрібно додати обгортку для цієї функції:

```
char* wrapper_factorial(int n){
  char* buf = malloc (sizeof (char) * 25);
  sprintf(buf, "%" PRIu64, factorial(n));
  return buf;
}
```

та ініціалізувати функцію в JavaScript.

```
let ffiFactorial = ffi('char* wrapper_factorial(int)');
```

Таблиця 2. Час виконання обчислень факторіалу

n	Час виконання в секундах			n!
	C	FFI	JS	
0	0.000014	0.00413	0.005113	1
1	0.000014	0.004128	0.005119	1
2	0.000014	0.004133	0.007452	2
3	0.000017	0.004137	0.00986	6
4	0.000018	0.004137	0.012238	24
5	0.000019	0.00414	0.014699	120
6	0.000019	0.004143	0.017425	720
7	0.00002	0.004146	0.020324	5040
8	0.000021	0.004152	0.023559	40320
9	0.000021	0.004165	0.026552	362880
10	0.000021	0.004199	0.029066	3628800
11	0.000023	0.004269	0.031581	39916800
12	0.000023	0.004215	0.034143	479001600
13	0.000024	0.004285	0.036445	6227020800
14	0.000025	0.004233	0.038926	87178291200
15	0.000025	0.004258	0.041334	1.30767E+12
16	0.000025	0.00433	0.044185	2.09228E+13
17	0.000026	0.004281	0.04765	3.55687E+14
18	0.000027	0.004361	0.051074	6.40237E+15
19	0.000027	0.004379	0.054181	1.21645E+17
20	0.000028	0.004329	0.056524	2.4329E+18

З результатів одразу видно, наскільки швидко була виконана функція на C.

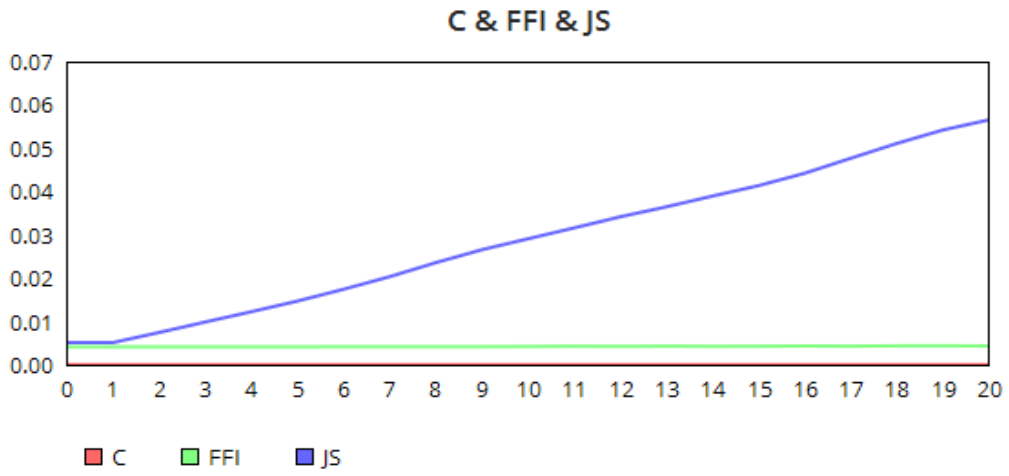

Рис. 6. Витрачений час на розрахунок n! для C, JavaScritpt та FFI.

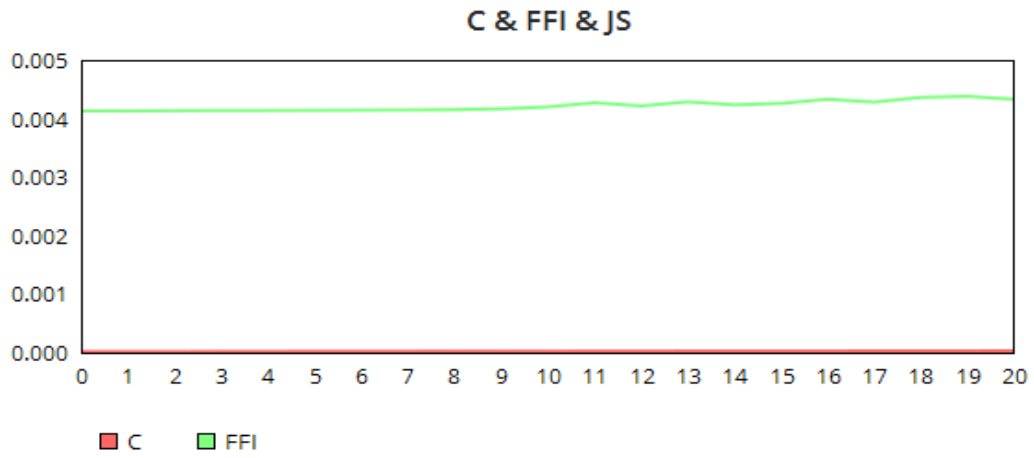

Рис. 7 — Витрачений час на розрахунок n! для C та FFI.

Також знову підтверджується здогадка про час виклику функції за допомогою FFI. Проте зараз середня різниця між C та FFI трохи зросла, та складає 0.004195, і це викликано додатковим кодом в обгортці.

Третя програма — розрахунок чисел Фібоначчі.

Код на C:

```
uint64_t fib(int n)
{
	uint64_t x = 1;
	uint64_t y = 0;
	for (int i = 0; i < n; i++)
```

```
    {
        x += y;
        y = x - y;
    }
    return y;
    }
```

Код на Javascript:

```
function fib(n) {
    let x = 1;
    let y = 0;
    for (let i = 0; i < n; i++)
    {
        x += y;
        y = x - y;
    }
    return y;
    }
```

Та використання обгортка для використання FFI:

```
char* wrapper_fib(int n){
  char* buf = malloc (sizeof (char) * 25);
  sprintf(buf, "%" PRIu64, fib(n));
  return buf;
}
```

Таблиця 3. Час виконання обчислень n-го числа Фібоначчі

n	Час виконання в секундах			n-е число Фібоначчі
	C	FFI	JS	
5	0.00001	0.003809	0.022597	5
10	0.00001	0.003833	0.036369	55
15	0.000011	0.003827	0.050095	610
20	0.000012	0.003832	0.06381	6765
25	0.000012	0.003851	0.077525	75025
30	0.000012	0.003859	0.091244	832040
35	0.000013	0.003856	0.10496	9227465
40	0.000013	0.003833	0.118674	102334155
45	0.000014	0.003895	0.132389	1134903170
50	0.000014	0.003887	0.146103	12586269025
55	0.000014	0.003903	0.159816	1.39584E+11
60	0.000015	0.003875	0.17353	1.54801E+12

65	0.000015	0.003929	0.187243	1.71677E+13
70	0.000015	0.003939	0.200955	1.90392E+14
75	0.000016	0.003961	0.214667	2.11149E+15
80	0.000017	0.00393	0.228384	2.34167E+16
85	0.000017	0.003997	0.2421	2.59695E+17
90	0.000018	0.003995	0.255812	2.88007E+18

Знову бачимо схожу картину: середній час, який потрібний для виклику FFI складає 0.003876. JavaScript значно поступається місцем у швидкості виконання як і C так і FFI.

Висновки. У статті порівняно способи реалізації програмного забезпечення для IoT на базі Mongoose OS та показано що для більш швидкого старту розробки можна залишити оптимізацію коду на потім, а в першу чергу швидко створити прототип, використовуючи JavaScript як мову сценаріїв, а при необхідності обчислень використати FFI, для комбінування JavaScript та C, це дає розробнику можливість використовувати простоту написання коду та одночасно зберегти швидкість виконання коду.

Список літератури

1. Mongoose OS - an IoT Firmware Development Framework [Електронний ресурс] — Режим доступу: https://github.com/cesanta/mongoose-os/ — (дата звертання 19.09.2019);
2. Mongoose OS - reduce IoT firmware development time up to 90% [Електронний ресурс] — Режим доступу: https://mongoose-os.com/ — (дата звертання 03.10.2019);
3. Mongoose OS documentation [Електронний ресурс] — Режим доступу: https://github.com/cesanta/mongoose-os-docs/ — (дата звертання 03.10.2019);
4. ESP speed compare in LUA, MicroPython, Arduino [Електронний ресурс] — Режим доступу: http://esp8266freq.blogspot.com/2016/12/esp-speed-compare-in-lua-micropython.html — (дата звертання 11.11.2019).