https://doi.org/10.31649/1997-9266-2022-164-5-80-85

УДК 621.396.6

А. В. Бруско[1]
О. Ю. Мирончук[1]

ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ БАГАТОЗАДАЧНОСТІ НА ПЛАТФОРМАХ RASPBERRY PI ТА ARDUINO

[1]Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Найпопулярнішими платформами для розробників вбудованих систем серед початківців є Arduino та Raspberry Pi. Ці платформи різні та мають різне призначення, проте на обох може виникнути потреба виконувати кілька операцій паралельно. В роботі розглянуто способи реалізації багатозадачності на вищезазначених платформах. Оскільки Arduino базуються здебільшого на одноядерних AVR мікроконтролерах, з досить низькою частотою тактування, то багатозадачність у цієї платформи лише умовна. Тривалі в часі операції будуть займати єдине ядро та єдиний потік мікроконтроллера, що не дозволить виконувати щось інше. Це не стосується апаратних блоків мікроконтроллера, таких як таймери, що працюють незалежно від програмної частини і дозволяють реалізувати виконання коротких завдань з фіксованим періодом. Для простого написання програми, що використовує такі можливості контролера, існують бібліотеки. Найвідомішою з них є TimedAction, що може бути знайдена на офіційному сайті Arduino. Також у більшості мікроконтролерів є апаратний вхід, за допомогою якого можна створювати переривання. Під час переривання також можна виконувати операції, після завершення яких буде відновлена робота головного циклу програми. На Raspberry Pi, як і на подібних мікрокомп'ютерах під керуванням операційної системи Linux, повноцінно можна розглянути потоки та процеси. Процес є екземпляром програми та створюється операційною системою. Окремі процеси не мають спільного об'єму пам'яті. Паралельні процеси використовуються, коли потрібно прискорити складні обчислення. Потоки є підзадачами, яких може бути декілька усередині процесу. Між кількома потоками є спільний об'єм пам'яті, що дозволяє зручно обмінюватись даними. Паралельні потоки використовують, коли потрібно виконувати тривалі операції вводу/виводу. У мові Python, що широко використовується у вбудованих системах, для реалізації багатопоточності використовують бібліотеку Threading. Особливістю багатозадачності у Python є Global Interpreter Lock. Цей механізм гарантує, що в один момент часу буде виконуватись код лише одного процесу програми. У цієї особливості Python є прихильники та противники, як відповідно переваги та недоліки.

Ключові слова: embedded, Arduino, Raspberry Pi, Python, мікроконтролер, мікрокомп'ютер, вбудовані системи, багатозадачність, процес, потік.

Вступ

Сучасні радіоелектронні пристрої, від простих до складних, часто містять елементи багатозадачності та паралелізму. Потреба багатозадачності виникає, коли необхідно одночасно виконувати дві чи більше операції, наприклад, очікувати ввід даних та оновлювати інформацію, що відображається. На сьогоднішній день широкого застосування у різних сферах набули мікрокомп'ютери Raspberry Pi. Здатність паралельно виконувати різні завдання дозволяє застосовувати Raspberry Pi для запуску сервера, обробки зображень, потокової передачі даних, розгортання нейронної мережі тощо [1]—[7]. Цю платформу доцільно використовувати, коли потужність персонального комп'ютера надлишкова, є обмеження габаритів або вартості пристрою.

Не менш популярною для розробників, особливо для новачків, на сьогодні є платформа Arduino. Обчислювальні потужності цієї платформи значно менші, ніж у попередньої, але й ціна також. За допомогою модулів розширення на платформі Arduino також можна розгорнути примітивний сервер, приймати, обробляти та передавати дані. Arduino дозволяє швидко реалізувати прототипи пристроїв, здебільшого інформаційних або автоматизаційних. А отже є випадки, коли без виконання кількох завдань паралельно просто не обійтись.

Метою статті є показати початківцям способи реалізації багатозадачності на двох найпопулярніших embedded платформах — Arduino (одноядерні мікроконтролери AVR) та Raspberry Pi (або інші сумісні комп'ютери з операційною системою Linux).

Платформа Arduino

Насамперед, справжню багатозадачність на одноядерних мікроконтроллерах не вдасться реалізувати. На відміну від них, відносно старі процесори Intel з технологією HyperThreading мали одне фізичне ядро та два віртуальних, що поділяли ресурси та діяли незалежно [8]. Запуск RTOS (Real Time Operating System) на мікроконтроллерах дозволяє наблизитись до імітації багатозадачності, але залишаються обмеження. Однак існують засоби, які дозволять виконувати певні операції паралельно. За правильної реалізації, такі операції не будуть заважати виконанню одна другій.

Платформа Arduino відома великою спільнотою, що активно працює над вдосконаленням бібліотек, спрощуючи реалізацію коду для початківців та прискорюючи її для досвідчених. TimedAction — одна з таких бібліотек. TimedAction створена для того, щоб приховати з головного файлу проекту реалізацію виконання періодичних завдань. В її основі лежить використання функції millis() лічильника, що починає роботу разом з подачею напруги живлення.

Суть роботи полягає в тому, що спочатку створюється об'єкт TimedAction, в який передаються значення приблизного періоду виконання, а також функція, яку потрібно виконати. Об'єкт містить такі методи:

– check() — виконання вказаної функції, якщо настала її черга;
– setInterval() — зміна періоду виконання;
– enable()/disable() — увімкнення та вимкнення певного завдання відповідно;
– reset() — скидання таймера певного завдання.

Алгоритм реалізації багатозадачності на прикладі виконання періодичних завдань з певним інтервалом ппоказано на рис. 1. Незалежний таймер, виконавши відлік часу, створює переривання, після якого починається виконання завдання. Після завершення виконання завдання мікроконтролер готовий відреагувати на наступне переривання, що змусить виконати вже інше завдання. Але якщо до моменту наступного переривання попереднє завдання не було завершене, то до виконання наступного мікроконтролер перейде не одразу.

Цим пояснюється неможливість справжньої багатозадачності на одноядерних мікроконтролерах. Проте це не означає, що контролер не може робити кілька завдань одночасно, а саме тих, що реалізуються апаратно. Наприклад, це робота таймерів та зовнішніх переривань.

У складі популярних моделей Arduino, в основі яких лежить мікроконтролер ATMEGA 328p, є 3 таймери, які налаштовуються окремо, а також зовнішні переривання, що налаштовуються індивідуально на кожен з двох виводів, або спільно на цілий регістр. Таймери нумеруються від 0 до 2, мають розрядність 8 і 16 біт та можуть працювати в таких режимах [9]:

– Normal — інкрементальний рахунок;

– Fast PWM — генерація сигналу з широтно-імпульсною модуляцією, за якої таймер виконує підрахунок в напрямку зростання і у разі переповнення лічильного регістра починає рахувати знову з нуля (логічний рівень на під'єднаному до таймера виводі змінюється на протилежний з

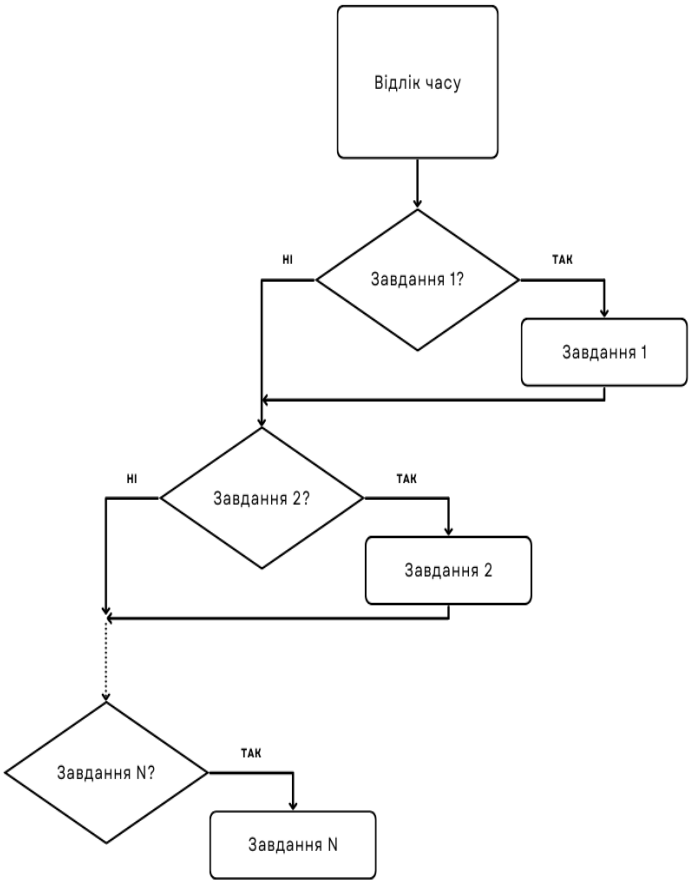

Рис. 1. Алгоритм реалізації багатозатачності на Arduino

досягненням лічильним регістром заданого значення та нуля);

– Phase Correct PWM — генерація сигналу з широтно-імпульсною модуляцією, коли таймер виконує підрахунок у напрямку зростання і у разі переповнення лічильного регістра переходить в підрахунок у напрямку спадання (логічний рівень на під'єднаному до таймера виводі змінюється на протилежний тільки з досягненням лічильним регістром заданого значення);

– CTC — скидання таймера у разі досягнення лічильним регістром заданого значення.

Кожен таймер може генерувати переривання, що призупинить виконання основного циклу, щоб виконати запрограмовану дію. Аналогічним чином працюють зовнішні переривання. Таким чином задачу вимірювання відліків часу, чи перевірки натискання на кнопку, можна доручити апаратній частині.

Для налаштування таймера в режимі генерації переривання із заданим часовим інтервалом необхідно розрахувати значення, яке необхідно записати в регістр порівняння. Цей розрахунок виконується за таким виразом:

$$CNT = \frac{T_{delay}}{T_{clk}} - 1 = T_{delay} \cdot F_{clk} - 1, \quad (1)$$

де CNT — значення регістру порівняння; T_{delay} — часовий інтервал, який необхідно забезпечити; T_{clk} — період тактування таймера; F_{clk} — частота тактування таймера.

Для прикладу, виконаємо розрахунок значення регістра порівняння, щоб забезпечити генерацію переривань 16 бітним таймером 1 з періодом 0,1 с за тактової частоти мікроконтролера 16 МГц

$$CNT = T_{delay} \cdot F_{clk} - 1 = 0{,}1 \cdot 16 \cdot 10^6 - 1 = 1599999.$$

Оскільки отримане значення перевищує максимально можливе, яке може бути записане в 16-розрядний регістр порівняння ($2^{16} - 1 = 65535$), необхідно скористатися подільником частоти тактування таймера, який може набувати значення 8, 64, 256, 1024. Вираз (1) з урахуванням значення подільника частоти набуде вигляду

$$CNT = \frac{T_{delay}}{T_{clk} \cdot DIV} - 1 = T_{delay} \cdot \frac{F_{clk}}{DIV} - 1, \quad (2)$$

де DIV — значення подільника частоти.

Розрахуємо значення регістра порівняння з урахуванням значення подільника частоти, рівним 64, за допомогою отриманого виразу (2)

$$CNT = T_{delay} \cdot \frac{F_{clk}}{DIV} - 1 = 0{,}1 \cdot \frac{16 \cdot 10^6}{64} - 1 = 24999.$$

Як видно з розрахунків, отримане значення регістра порівняння не перевищує максимально можливого. Таким чином, відлік заданого часового інтервалу може бути реалізованим за допомогою вибраного таймера.

Слід зазначити, що важливу роль під час виконання паралельних операції відіграє синхронізація мікроконтролера з зовнішніми периферійними пристроями, задіяними в цих операціях. Для цього використовується здатність виводити сигнали переривання на порти виводу мікроконтролера.

Таким чином можна зазначити, що хоч і неможливо програмно реалізувати багатозадачність мікроконтролера, проте є можливість виконувати паралельні операції, які займатимуть ядро лише у потрібний момент. За грамотного використання апаратних можливостей паралельні задачі не заважатимуть роботі один одного і під час користування не будуть створюватись небажані затримки.

Платформа Raspberry Pi

Raspberry Pi є одним з найпопулярніших на сьогодні мікрокомп'ютером, популярним для використання як у професійних, так і в аматорських розробках. У порівнянні з Arduino мікрокомп'ютер є складною системою, у якій програма користувача вже буде не єдиним процесом. Raspberry Pi має багато аналогів на ринку, що відрізняються як процесором, так і периферією, але всіх їх об'єднує операційна система Linux [10]—[12]. Застосування цієї операційної системи дає змогу запускати ту саму програму на різних платах, не переписуючи її повністю.

Реалізацію багатозадачності розглянемо у контексті застосування мови програмування Python для написання програм [13], [14]. Для цього слід розглянути два поняття: процес та потік. Процес — це екземпляр програми. Процеси створюють потоки (підпотоки) для виконання підзадач, як, до прикладу, зчитування клавіатури, завантаження сторінок, збереження файлів. Потоки існують всередині процесів і використовують спільний об'єм оперативної пам'яті, що показано на рис. 2.

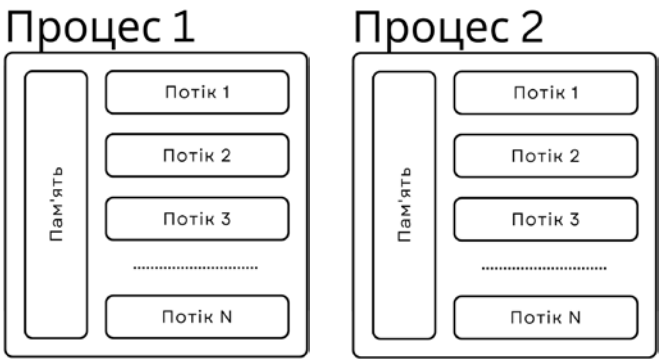

Рис. 2. Процеси і потоки

Як приклад, розглянемо програму текстового редактора. Коли користувач відкриває редактор, то створюється процес. Коли користувач починає друкувати, процес створює потоки: один — для зчитування клавіатури, другий — для відображення тексту, третій — для автозбереження файлу та ще один — для розпізнавання і виділення помилок. Створюючи кілька потоків, редактор використовує простій центрального процесора (очікування вводу символів, або вивантаження файлу) та не спричиняє додаткових затримок.

Процеси:

- Створюються операційною системою для запуску програм;
- Можуть містити декілька потоків;
- Кілька процесів можуть виконувати код одночасно в одній програмі Python;
- Використовувати процеси для операцій вводу/виводу накладніше, оскільки це вимагає більше часу;
- Обмін даними між процесами повільніший, ніж з потоками, оскільки процеси не мають спільного об'єму оперативної пам'яті. У Python вони обмінюються даними за допомогою структур даних, таких як масиви, що вимагає затрат часу на операції вводу/виводу.

Потоки:

- Є складовими процесів;
- Мають спільний об'єм пам'яті і ефективний доступ до спільних змінних;
- Два потоки не можуть виконувати код одночасно в одній програмі Python (однак є способи обходу обмеження).

Центральний процесор виконує базові обчислення у комп'ютері. Процесори містять одне, або декілька ядер, що дозволяють виконувати програми одночасно. З одним ядром неможливо досягти прискорення у задачах з інтенсивним використанням центрального процесора. Операційна система постійно переключатиметься між завданнями на короткий проміжок часу. Саме тому з дрібними задачами (наприклад, завантаження кількох зображень) багатозадачність може деколи зашкодити продуктивності. Тому існує ускладнення, пов'язане із запуском та роботою кількох задач на одному ядрі.

Про доцільність використання потоків і процесів можна зробити таке узагальнення:

– процеси прискорюють роботу програми на Python, якщо виконуються складні обчислення. Таким чином використовується перевага багатоядерних центральних процесорів і унеможливлюється GIL (Global Interpreter Lock);

– потоки доцільніше використовувати для операцій вводу/виводу, що включають зовнішні системи, файли, тощо. Це також спрощує передачу даних усередині програми;

– потоки не дають переваги у швидкодії, якщо виконувати у них паралельні обчислення, оскільки GIL блокує їх одночасне виконання.

Висновки

Неможливо реалізувати справжню паралельність задач суто програмно, тобто без допомоги апаратної реалізації (багатоядерна система, таймери, тригери переривань). Будь-яка тривала в часі задача не дозволятиме коректно виконувати інші завдання, що викликаються частіше. Проте, за наявності передбаченого механізму поділу на ядра чи потоки, система може виконувати паралельно та одночасно декілька незалежних завдань.

У випадку Arduino, розробнику пропонується планувати виконання окремих завдань, таким чи-

ном, щоби їхнє послідовне виконання не створювало затримок. Розробникам на Raspberry Pi та на аналогічних платформах потрібно вибрати такий поділ програми на потоки і процеси, який забезпечить найпродуктивніше виконання конкретної задачі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] J. Marot, and S. Bourennane, "Raspberry Pi for image processing education," *2017 25th European Signal Processing Conference* (EUSIPCO), 2017, pp. 2364-2366, https://doi.org/10.23919/EUSIPCO.2017.8081633 .

[2] V. Bharadwaja, R. Ananmy, S. Nikhil, K. V. Vineetha, J. Shah, and D. G. Kurup, "Implementation of Artificial Neural Network on Raspberry Pi for Signal Processing Applications," in *2018 International Conference on Advances in Computing, Communications and Informatics* (ICACCI), 2018, pp. 1488-1491, https://doi.org/10.1109/ICACCI.2018.8554887 .

[3] A. P. Jadhav, and V. B. Malode, "Raspberry PI Based OFFLINE MEDIA SERVER," *2019 3rd International Conference on Computing Methodologies and Communication* (ICCMC), 2019, pp. 531-533, https://doi.org/10.1109/ICCMC.2019.8819718 .

[4] D. Eridani, and E. D. Widianto, "Performance of Sensors Monitoring System using Raspberry Pi through MQTT Protocol," in *2018 International Seminar on Research of Information Technology and Intelligent Systems* (ISRITI), 2018, pp. 587-590, https://doi.org/10.1109/ISRITI.2018.8864473 .

[5] А. В. Бруско, «Інтерактивний логотип,» в *матер. IV Всеукраїнської науково-технічної конференції студентів та аспірантів «Радіоелектроніка у XXI столітті»*, 25-26 травня 2021 р., Київ, Україна, КПІ ім. Ігоря Сікорського, РТФ. Київ : КПІ ім. Ігоря Сікорського, 2021, с. 19-20.

[6] С. Б. Могильний, Мікрокомп'ютер Raspberry Pi — інструмент дослідника. Київ, Україна: Талком, 2014. 340 с.

[7] О. Ю. Мирончук, О. О. Шпилька, Д. Д. Струков, А. А. Петровський, і А. О. Герасименко, «Застосування нейронної мережі для оцінювання частотної характеристики багатопроменевого каналу в системах зв'язку з технологією OFDM,» Вісник Вінницького політехнічного інституту, № 4, с. 99-104, Серп. 2021.

[8] Xinmin Tian, Yen-Kuang Chen, M. Girkar, S. Ge, R. Lienhart, and S. Shah, "Exploring the use of Hyper-Threading technology for multimedia applications with Intel/spl reg/ OpenMP compiler," *Proceedings International Parallel and Distributed Processing Symposium*, 2003, pp. 8. https://doi.org/ 10.1109/IPDPS.2003.1213118 .

[9] H.-W. Huang, The Atmel AVR Microcontroller: MEGA and XMEGA in Assembly and C. Cengage Learning, 2013, 816 p.

[10] D. Bovet, and M. Cesati, *Understanding the Linux Kernel*, Janaury 2001. OŔeally & Associates, Sebastopol (2001)

[11] B. J. Catanzaro, *Multiprocessor system architectures.* Englewood Cliffs, N.J : PTR Prentice Hall, 1994.

[12] G. P. Duggan, and P. Young, "Precision timing on low-cost Linux microcomputers," in *2015 Annual IEEE Systems Conference (SysCon) Proceedings*, 2015, pp. 469-471, https://doi.org/ 10.1109/SYSCON.2015.7116795 .

[13] A. Martelli, and D. Ascher, Eds., 2002. *Python Cookbook.* Sebastopol: O'Reilly. ISBN: 0-596-00167-3 .

[14] D. Saito, H. Washizaki, Y. Fukazawa, T. Yoshida, I. Kaneko, and H. Kamo, "Learning Effects in Programming Learning Using Python and Raspberry Pi: Case Study with Elementary School Students," in *2019 IEEE International Conference on Engineering, Technology and Education* (TALE), 2019, pp. 1-8, https://doi.org/10.1109/TALE48000.2019.9225866 .

Рекомендована кафедрою інформаційних радіоелектронних технологій і систем ВНТУ

Стаття надійшла до редакції 21.10.2022

Бруско Андрій Вадимович — студент радіотехнічного факультету, e-mail: andrewbrusko@gmail.com ;

Мирончук Олександр Юрійович — PhD, старший викладач кафедри радіотехнічних систем, e-mail: myronchukalex@gmail.com .

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ

A. V. Brusko[1]
O. Yu. Myronchuk[1]

Features of the Implementation of Multitasking on the Raspberry Pi and Arduino Platforms

[1]National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

The most popular platforms for developers of embedded systems among beginners are Arduino and Raspberry Pi. These platforms are different and have different purposes, but both may need to perform multiple operations in parallel. The article shows ways to implement multitasking on the above-mentioned platforms. Since Arduinos are mostly based

on single-core AVR microcontrollers, with a rather low clock frequency, the multitasking of this platform is only conditional. Time-consuming operations will occupy the only core and the only thread of the microcontroller, which will not allow doing anything else. This does not apply to the hardware components of the microcontroller, such as timers, which work independently from the software part and allow the implementation of short tasks with a fixed period. There are libraries for simply writing a program that uses such controller capabilities. The most famous of them is TimedAction, which can be found on the official Arduino website. Also, most microcontrollers have a hardware input that can be used to generate software interrupts. During the interruption, operations can also be performed, after the completion of which the work of the main loop of the program will be resumed. On the Raspberry Pi, as on similar microcomputers running the Linux OS, you can fully consider threads and processes. A process is an instance of a program and is created by the operating system. Individual processes do not share a common memory. Parallel processes are used when complex calculations need to be accelerated. Threads are subtasks, which can be several within a process. There is a common amount of memory between several threads, which allows convenient data exchange. Parallel threads are used when you need to perform long I/O operations. In the Python language, which is widely used in embedded systems, the Threading library is used to implement multithreading. A feature of multitasking in Python is the Global Interpreter Lock. This mechanism ensures that the code of only one application process will be executed at a time. This feature of Python has supporters and detractors, as well as advantages and disadvantages.

Keywords: embedded, Arduino, Raspberry Pi, Python, microcontroller, microcomputer, embedded systems, multitasking, process, thread.

Brusko Andrii V. — Student of the of the Department of Radio Engineering, e-mail: andrewbrusko@gmail.com ;
Myronchuk Oleksandr Yu. — PhD, Senior Lecturer of Chair of Radio Engineering Systems, e-mail: myronchukalex@gmail.com