Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Especialidad en Sistemas Embebidos

A Thread-Safe Library for Concurrent Web Service Request Handling

---

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: **GUSTAVO FLORES GARCÍA**

Tutor **SERGIO NICOLÁS SANTANA SÁNCHEZ**

Tlaquepaque, Jalisco. Agosto de 2025

A Thread-Safe Library for Concurrent Web Service Request Handling

Gustavo Flores García
Departamento de Electrónica, Sistemas e Informática
Instituto Tecnológico y de Estudios Superiores de Occidente
Tlaquepaque, México
gflores.garcia@iteso.mx

Sergio Nicolás Santana Sánchez
Departamento de Electrónica, Sistemas e Informática
Instituto Tecnológico y de Estudios Superiores de Occidente
Tlaquepaque, México
sergioni@iteso.mx

***Abstract* — IoT devices require the incorporation of low-level software components during complex integration processes within multithreaded environments. The aim of this work is to design a thread-safe library to consume web services by establishing secure connections to multiple servers. The thread-safe library integrates embedded software components such as LwIP and Mbed TLS, and exposes an Application Programming Interface (API) to establish multiple connections to the Particle IoT cloud via REST API web service calls. The embedded application is developed on the NXP RW612 microcontroller and includes a graphical user interface (GUI) to facilitate interaction with the end user. Thread-safe library demonstrated that the unification of low-level components is viable and provided a solution to reduce the complex integration processes in resource-constrained environments. This work serves as a reference for future studies to analyze other cryptographic libraries, such as wolfSSL and BearSSL, and their compatibility in multithreaded environments, including specific hardware resources like hardware cryptographic accelerators.**

Keywords — Thread-safe library, Mbed TLS, HTTP, REST API, RTOS, multitasking, embedded application

I. INTRODUCTION

The accelerated growth in the number of IoT devices in recent years has highlighted the significance of secure communication channels to maintain trust, protect privacy, and ensure the secure operation of IoT devices [1]. Hypertext Transfer Protocol Secure (HTTPS) is an encrypted communication protocol that combines the Hypertext Transfer Protocol (HTTP) with Transport Layer Security (TLS) by first establishing a secure connection and transferring encrypted data to ensure confidentiality, integrity and authentication between the server and client. In fact, there is an open-source implementation of the TLS protocol called Mbed TLS [2]. Furthermore, HTTP is generally used by Representational State Transfer Application Programming Interface (REST API), which exposes multiple resources and each resource is represented as a unique Uniform Resource Location (URL) with different HTTP methods [3].

Mbed TLS is a cryptographic library optimized for embedded systems with limited hardware resources. This library provides efficient functions to execute several processes such as handshake TLS, handling of X.509 digital certificates and encryption/decryption of data transmitted between the client and server. Mbed TLS also supports threaded environments, making it an excellent choice for implementing multitasking systems with Real-Time Operating Systems (RTOS). RTOS enables efficient multitasking in embedded systems, therefore thread safety is essential when designing libraries that may be accessed simultaneously by multiple threads.

To establish multiple secure connections, Mbed TLS provides TLS-based encryption. The embedded systems that require a secure connection to access resources from remote servers must integrate multiple low-level components such as stack TCP/IP, TLS-based encryption and RTOS. As a result, developers may encounter considerable complexity during the integration process. Therefore, an intermediate layer that unifies the interaction between the system's low-level components is required.

This paper presents the design of a thread-safe library as an alternative solution to build HTTPS requests and consume web services from a REST API by establishing secure connections to multiple servers.

The remainder of the paper is organized as follows: Section II presents an overview of secure communication protocols and related libraries used in this work. Section III describes the methodology used to design the developed thread-safe library. Section IV explains the implementation in an embedded application. Section V analyzes the results of the implementation. Finally, Section VI concludes this paper and discusses future improvements in the thread-safe library.

II. BACKGROUND

This implementation incorporates specific embedded software components that align with the demands of resource-constrained systems. The thread-safe library requires secure web service consumption, therefore the essential technologies involved in secure communication for embedded systems include the use of HTTP as a communication protocol between clients and servers and the fundamental function of TLS in encrypted channels. The embedded software components, such as Lightweight TCP/IP (LwIP) [4] and Mbed TLS set the stage for secure communication in embedded applications. In the context of IoT, these elements play a critical role in enabling embedded systems to operate within IoT environments.

A. Internet of Things (IoT)

IoT devices are composed of hardware and software components that facilitate internet connectivity. These devices, often implemented as embedded systems, are designed to transmit and receive information from cloud services. To date, IoT has significantly increased the number of connected devices with approximately 19.8 billion worldwide [5].

B. *Hypertext Transfer Protocol Secure (HTTPS)*

HTTP application-layer protocol is a fundamental option to integrate within the thread-safe library, due to its role in providing the foundation to establish communication between two applications; this characteristic allows access, sending, and receiving of information through methods such as GET, POST, PUT, and DELETE, which is fundamental for consuming web services. Additionally, HTTP defines status codes that indicate the results of the operation and facilitate error handling.

Subsequently, HTTPS is an enhancement of HTTP that enables a secure communication channel over TLS connection. HTTPS provides confidentiality and authenticity, which guarantees the integrity of data transmitted over the network [6].

C. *Transport Layer Security (TLS)*

The use of the TLS protocol is a key element within the thread-safe library, as establishing secure connections in IoT environments is critical for protection of information integrity. By TLS protocol integration, the thread-safe library ensures that web services can be consumed through an encrypted channel.

Furthermore, TLS involves two main layers: 1) TLS Record Protocol provides confidentiality and reliable connection security and 2) TLS Handshake Protocol enables authentication between the client and server [7]. The Handshake Protocol ensures three key properties:

- Peer identity authentication using asymmetric or public key cryptography.
- Secure negotiation of a shared secret.
- Reliable negotiation process.

D. *Embedded Software Components*

The embedded software components described below provide the essentials resources to ensure that an embedded application can operate reliably within an IoT ecosystem. Designed specifically for use in resource-constrained systems, they represent an excellent option for integration within the thread-safe library.

- LwIP: Lightweight TCP/IP is a compact and efficient implementation of the TCP/IP protocols, specifically developed to enable IP networking capabilities in embedded systems. This implementation is essential to provide network-level communication to resource-constrained systems.

- Mbed TLS: Mbed TLS provides TLS capabilities for embedded systems, ensuring a secure communication channel with the portable implementations of secure communication protocols. This embedded software component offers compatibility with hardware cryptographic accelerators.

III. THREAD-SAFE LIBRARY DESIGN

A. *Thread-Safe Mechanism*

A critical aspect for developing reliable and efficient embedded systems is thread safety. This concept provides thread-safe access capability to execute multiple threads that can access shared code. Thread safety includes synchronization and protection mechanisms; among the most common mechanisms are mutex, semaphores, read-write lock, messages queues and condition variables.

B. *Design Overview*

The proposed library is designed to provide a robust API to consume web services from multiple servers. This library involves specific versions of complex embedded software components such as Mbed TLS 2.28.8, LwIP 2.2.1 and FreeRTOS Kernel V11.0.1.

As illustrated in “Fig. 1”, the block diagram represents the software stack including the elements mentioned previously. The library unifies these elements by providing functions to reduce integration complexity. Each element of the software stack plays an essential role in the design as described below:

- Application Code: Uses the thread-safe library.
- Mbed TLS: Provides an abstraction layer for secure communication.
- LwIP: Supports an interface for network communication.
- FreeRTOS: Enables capability for multitasking and thread safety.
- Microcontroller: Supplies the processor core, memory and peripherals.

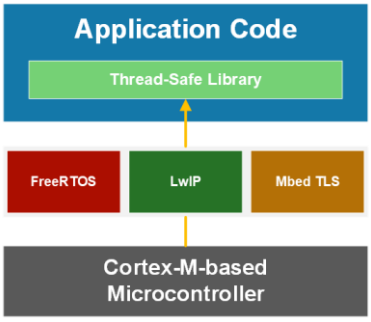

Fig. 1. Block diagram of the software stack.

The thread-safe library consists of three fundamental files, as described below:

- Configuration file *request_https_cfg.h*: Defines macros and compilation directives to configure parameters and manage input validations within the implemented functions.
- Header file *request_https.h*: Declares the public functions and data structures, serving as the interface between the thread-safe library and the embedded application.
- Source file *request_https.c*: Contains the implementation of the declared functions and manages the internal operational logic.

In the context of thread-safe mechanisms, it is relevant to define the types of variables that the thread-safe library manages internally and to determine whether additional thread-safe mechanisms are needed to protect shared resources or variables.

The types of variables used in the C programming language are described below:

- Local variables: These exist only during the execution of a function. Each function call generates a new and independent context; therefore, local variables ensure safe operation in multithreaded environments.
- Global variables: These persist in memory until the program finishes and can be accessed by any function of the program. Proper protection mechanisms are required to ensure safe concurrent access.

Considering the characteristics of local variables, “Fig. 2” illustrates their implementation within the thread-safe library. Consequently, proper validation of their integration in multithreaded environments is required.

```
154     /* LWIP structures and variables */
155     struct addrinfo hints;
156     struct addrinfo *ai;
157     struct sockaddr_in server;
158     int sock;
159     int result = 1;
160
161     #if HTTPS
162         /* Mbed TLS context */
163         int res;
164         mbedtls_ssl_context ssl;
165         mbedtls_ssl_config conf;
166         mbedtls_entropy_context entropy;
167         mbedtls_ctr_drbg_context ctr_drbg;
168         mbedtls_x509_crt cacert;
169     #endif
```

Fig. 2. Implementation of local variables within the thread-safe library.

C. *Integration with RTOS*

A significant advantage in the integration process is that Mbed TLS provides specific file configurations that allow compatibility with RTOS. For this integration, FreeRTOS Kernel V11.0.1 served as the real-time operating system. The configuration to enable Mbed TLS in threaded environments is detailed below.

“Fig. 3” illustrates the macro required to enable the threading abstraction layer. This configuration must be applied in the *mbedtls_config.h* file. When a hardware cryptographic accelerator is available, the EdgeLock Secure Enclave (ELS) configuration is used, therefore the configuration must be applied on *els_pck_mbedtls_config.h* file.

```
3736  * This allows different threading implementations (self-implemented or
3737  * provided).
3738  *
3739  * You will have to enable either MBEDTLS_THREADING_ALT or
3740  * MBEDTLS_THREADING_PTHREAD.
3741  *
3742  * Enable this layer to allow use of mutexes within Mbed TLS
3743  */
3744 #define MBEDTLS_THREADING_C
```

Fig. 3. Preprocessor directive MBEDTLS_THREADING_C.

“Fig. 4” shows another configuration in the *els_pck_mbedtls_config.h* file to enable the implementation of alternative threading functions if required.

```
2376  * Requires: MBEDTLS_THREADING_C
2377  *
2378  * Uncomment this to allow your own alternate threading implementation.
2379  */
2380 #define MBEDTLS_THREADING_ALT
```

Fig. 4. Preprocessor directive MBEDTLS_THREADING_ALT.

These configurations are essential, as Mbed TLS does not provide a threading environment by default. To complete the setup, the *mbedtls/threading.h* file needs to be included in the thread-safe library and the mutex handling functions must be implemented, as illustrated in “Fig. 5”.

```
45  void mutex_init(mbedtls_threading_mutex_t *mutex)
46  {
47      mutex->mutex = xSemaphoreCreateMutex();
48      mutex->is_valid = (mutex->mutex != NULL);
49  }
50
51  void mutex_free(mbedtls_threading_mutex_t *mutex)
52  {
53      if (mutex->is_valid && mutex->mutex != NULL) {
54          vSemaphoreDelete(mutex->mutex);
55          mutex->is_valid = 0;
56      }
57  }
58
59  int mutex_lock(mbedtls_threading_mutex_t *mutex)
60  {
61      if (mutex->is_valid && xSemaphoreTake(mutex->mutex, portMAX_DELAY) == pdTRUE) {
62          return 0;
63      }
64      return MBEDTLS_ERR_THREADING_MUTEX_ERROR;
65  }
66
67  int mutex_unlock(mbedtls_threading_mutex_t *mutex)
68  {
69      if (mutex->is_valid && xSemaphoreGive(mutex->mutex) == pdTRUE) {
70          return 0;
71      }
72      return MBEDTLS_ERR_THREADING_MUTEX_ERROR;
73  }
```

Fig. 5. Implementation of mutex handling functions.

The following configuration depends on hardware features; as a result, the *MBEDTLS_SSL_MAX_CONTENT_LEN* parameter is critical when hardware resources are limited. Each secure connection requires a specific amount of heap memory; therefore, this setting defines the maximum size of the input/output buffer, as illustrated in “Fig. 6”.

```
305 #if !defined(MBEDTLS_SSL_MAX_CONTENT_LEN)
306 #define MBEDTLS_SSL_MAX_CONTENT_LEN         4096   /**< Size of the input / output buffer
307 #endif
```

Fig. 6. Definition of the macro MBEDTLS_SSL_MAX_CONTENT_LEN.

IV. Embedded Application Using the Thread-Safe Library

The embedded application is developed on a development board called FRDM-RW612 that integrates an NXP microcontroller RW612. This embedded application uses a thread-safe library designed to establish communication with the Particle IoT cloud REST API. The embedded application ecosystem is illustrated in “Fig. 7”.

Fig. 7. Embedded application as part of an IoT ecosystem.

A. *Particle IoT*

Particle IoT provides a REST API that permits developers to interact with connected devices. This REST API can be tested using the tool called Postman. “Fig. 8” shows the Postman environment and an example of web service consumption.

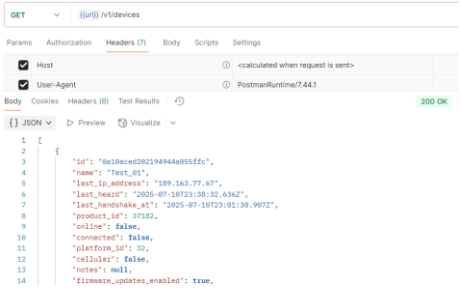

Fig. 8. Postman environment and web service usage.

The steps required to enable IoT Particle devices are described below:

1. Create a Particle account.
2. Configure the device using a web browser by accessing setup.particle.io.
3. Update firmware device.
4. Integrate with cloud services and download the first program.

The embedded application communicates with a Photon 2 development module [8], which is an IoT Particle device that serves as an end device. The application uses the thread-safe library to consume web services from Particle IoT cloud to request information or execute actions on the Photon 2.

Photon 2 provides Wi-Fi connectivity and integration with Particle cloud. To program a Particle device, access to Particle Web IDE is required, which allows the main functions such as write and edit code, compile firmware, flash firmware, library management and connect devices. “Fig. 9” shows a Particle Web IDE environment.

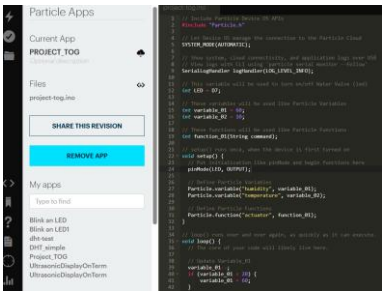

Fig. 9. Particle Web IDE environment and code implemented in the IoT Particle device.

B. *Embedded application using FRDM-RW612*

FRDM-RW612 [9] integrates an MCU RW612 [10] that has a network interface communication Wi-Fi 6, hardware cryptographic accelerator and provides 1.2 MB on-chip SRAM. These characteristics are essential for IoT device requirements.

The embedded application provides a Graphical User Interface (GUI) integrating a display touch 480x320 IPS TFT MCD Module that enables the interaction of the final users. GUI is designed using the GUI Guider software and is based on the open-source implementation Light and Versatile Graphics Library (LVGL), as illustrated in “Fig. 10”.

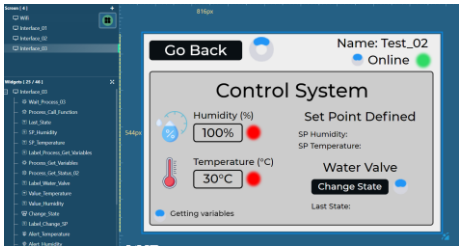

Fig. 10. GUI Guider environment and GUI designed for the embedded application.

The source code of the embedded application is developed in MCUXpresso IDE. This application uses FreeRTOS Kernel V11.0.1 to enable a threaded environment and create specific tasks to establish Wi-Fi connection, control the GUI and consume of multiple web services simultaneously. Each web service is handled by a dedicated task, which invokes the thread-safe library to create a secure connection instance. “Fig. 11” illustrates a web service consuming data from Particle IoT cloud REST API, incorporating functions provided by the thread-safe library.

```
PRINTF("Get Devices Process\r\n");

/* HTTP context initialization */
request_http_init(&conn_01.context);

/* Build HTTP request */
request_add_request_line("GET /v1/devices HTTP/1.1\r\n", &conn_01.context.elements);
request_add_header("Host: api.particle.io\r\n", &conn_01.context.elements);
request_add_header("Authorization: Bearer d7955bdf\r\n", &conn_01.context.elements);

/* Create HTTP request */
request_http_create(&conn_01.context.elements, &conn_01.context.request);

/* Connection parameters configuration */
set_conn_info(&conn_01.info, SERVER_NAME, (int)SERVER_PORT, certificate);

/* Call function to execute web service */
if (execute_web_service(&conn_01)) {
    PRINTF("HTTP request could be executed correctly\r\n");
}
else {
    PRINTF("There is an error during executing process\r\n");
}
```

Fig. 11. Web service consumption using functions of the thread-safe library.

V. RESULTS

The thread-safe library was developed to support multiple concurrent remote server connections in a multithreaded environment. The main objective was to ensure that the thread-safe library correctly handles multiple secure connection instances with concurrent access.

To validate the correct behavior of the thread-safe library, the SEGGER SystemView tool was employed. A test scenario was designed in which three independent tasks with the same priority level were created. Each task consumes a web service every 10 ms using the functions provided by the thread-safe library. As illustrated in “Fig. 12”, additional SystemView functions were used to ensure that the synchronization mechanisms operate as expected. In addition, a delay was implemented after each task acquires the mutex, intentionally causing the other tasks to compete for the same resource.

```
int mutex_lock(mbedtls_threading_mutex_t *mutex)
{
    SEGGER_SYSVIEW_OnUserStart(1);

    if (mutex->is_valid && xSemaphoreTake(mutex->mutex, portMAX_DELAY) == pdTRUE) {

        vTaskDelay(pdMS_TO_TICKS(10));

        return 0;
    }

    return MBEDTLS_ERR_THREADING_MUTEX_ERROR;
}

int mutex_unlock(mbedtls_threading_mutex_t *mutex)
{
    if (mutex->is_valid && xSemaphoreGive(mutex->mutex) == pdTRUE) {

        SEGGER_SYSVIEW_OnUserStop(1);

        return 0;
    }

    return MBEDTLS_ERR_THREADING_MUTEX_ERROR;
}
```

Fig. 12. Integration of SystemView functions to mark user-defined events.

“Fig. 13” illustrates the SEEGER SystemView environment, which visualizes the tasks executed within the RTOS. The trace

confirms that tasks wait for the mutex to be released before taking control, demonstrating that the thread-safe library's functions handle concurrent access correctly.

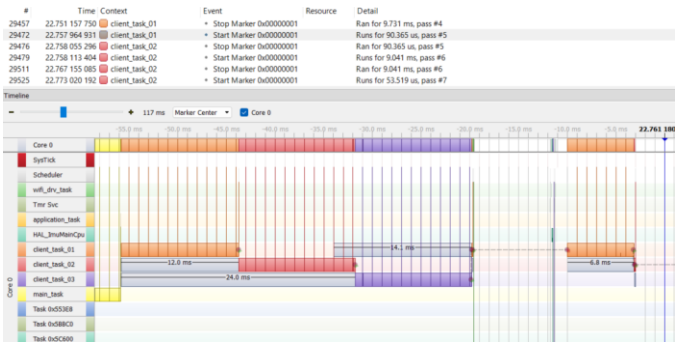

Fig. 13. SEGGER SystemView environment and analysis of tasks and synchronization mechanisms.

Additionally, tests confirmed that the thread-safe library correctly manages heap memory allocation. "Fig. 14" shows a code implemented to monitor and validate the heap memory consumption with three secure connections simultaneously. This test demonstrates heap memory usage of approximately 68,688 bytes (67.0781 KB); the amount of heap memory configured on the MCU RW612 was 256 KB.

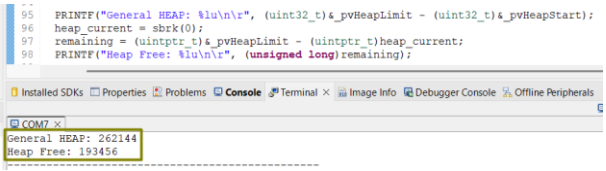

Fig. 14. Web service consumption using functions of the thread-safe library.

"Fig. 15" illustrates the embedded application operating successfully with the integrated thread-safe library. The GUI displays information retrieved from the Particle IoT cloud via REST API web service calls.

Fig. 15. Embedded application consuming web services from Particle IoT cloud REST API.

VI. Conclusion and Outlook

Integrating the developed thread-safe library into the embedded application enabled the reliable execution of multiple secure connections. The results confirm that unifying essential embedded software components reduces integration complexity in multithreaded contexts without exceeding the available memory resources, supporting scalability for resource-constrained systems.

Future work should explore the use of alternative open-source cryptographic libraries, including wolfSSL and BearSSL, to assess their performance and thread safety in multitasking scenarios compared to Mbed TLS. Additionally, examining their integration with hardware cryptographic accelerators could enhance processing efficiency.

References

[1] A. R. Alkhafajee, A. M. A. Al-Muqarm, A. H. Alwan, and Z. R. Mohammed, "Security and Performance Analysis of MQTT Protocol with TLS in IoT Networks," in *2021 4th International Iraqi Conference on Engineering Technology and Their Applications (IICETA)*, Najaf, Iraq: IEEE, Sept. 2021, pp. 206–211. doi: 10.1109/iiceta51758.2021.9717495.

[2] The Mbed TLS Contributors, "Mbed TLS documentation." Accessed: July 14, 2025. [Online]. Available: https://mbed-tls.readthedocs.io/en/latest/

[3] H. Garg and M. Dave, "Securing IoT Devices and SecurelyConnecting the Dots Using REST API and Middleware," in *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, Ghaziabad, India: IEEE, Apr. 2019, pp. 1–6. doi: 10.1109/IoT-SIU.2019.8777334.

[4] Free Software Foundation, Inc, "LwIP - A Lightweight TCP/IP stack." Accessed: July 14, 2025. [Online]. Available: https://savannah.nongnu.org/projects/lwip/

[5] Statista, "Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2034." Accessed: July 08, 2025. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[6] D. Caballero, F. Gonzalez, and S. A. Islam, "Analysis of Network Protocols for Secure Communication," in *2021 9th International Symposium on Digital Forensics and Security (ISDFS)*, Elazig, Turkey: IEEE, June 2021, pp. 1–6. doi: 10.1109/ISDFS52919.2021.9486356.

[7] J. Göppert, A. Chomel, J. S. E, and A. Sikora, "Performance Evaluation of TLS Session Establishments on an ARM Cortex-M4 Platform," in *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Dortmund, Germany: IEEE, Sept. 2023, pp. 1205–1210. doi: 10.1109/idaacs58523.2023.10348947.

[8] Particle, "Photon 2 Datasheet." Accessed: July 14, 2025. [Online]. Available: https://docs.particle.io/reference/datasheets/wi-fi/photon-2-datasheet/

[9] NXP Semiconductors, "FRDM-RW612 Development Board." Accessed: July 07, 2025. [Online]. Available: https://www.nxp.com/design/design-center/development-boards-and-designs/FRDM-RW612

[10] NXP Semiconductors, "RW612 Wireless MCU Datasheet." NXP Semiconductors, 2024. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/RW612.pdf