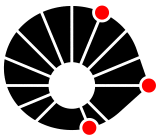

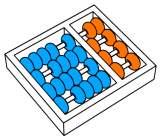

SmartLock Lite: um sistema de controle de acesso usando o microcontrolador ESP32

G. Lima

Technical Report - IC-22-06 - Relatório Técnico
November - 2022 - Novembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

SmartLock Lite: um sistema de controle de acesso usando o microcontrolador ESP32

Resumo

O ESP32 é um microcontrolador com interfaces de comunicação Wi-Fi e Bluetooth que vem sendo utilizado em diversos projetos de Internet das Coisas IoT devido a sua versatilidade. Aliado a outros componentes, pode ser usad o para implementar um sistema de controle de acesso de baixo custo e fácil implementação. Neste trabalho foi implementada uma versão do projeto SmartLock IoT[1], originalmente desenvolvido na plataforma de hardware Raspberry Pi, usando o ESP32, chamada de SmartLock Lite.

1 Introdução

O uso de sistemas embarcados para coletar dados e acionar atuadores abre possibilidades para desenvolver soluções de monitoramento e automação de ambientes. Tais sistemas, comumente implementados por meio de microcontroladores associados a interfaces de comunicação sem fio, possibilitam a conexão de microcontroladores à Internet oferecendo soluções para diversos problemas que envolvem interação com usuários e sistemas remotos. Adicionalmente, o avanço das tecnologias de hardware, dispositivos como o Raspberry Pi[16] permitem desenvolver projetos com funcionalidade de captação e processamento de imagens. Com isso em mente, o grupo SmartCampus UNICAMP[17] desenvolveu a SmartLock IoT, uma fechadura eletrônica conectada à Internet, que realiza controle de acesso a ambientes por meio de Cartões RFID[13], agendamento e leitura de QRCodes. a SmartLock IoT apresentou ótimos resultados em seu funcionamento, entretanto, o custo do hardware utilizado dificulta sua implantação em larga escala na Universidade. Visando reduzir o custo da solução, este trabalho teve como foco o desenvolvimento de uma solução semelhante usando um microcontrolador com menor capacidade de processamento, armazenamento e número de portas de entrada e saída digital e, consequentemente, mais barato. A nova versão desenvolvida foi denominada SmartLock Lite. O restante deste documento descreve as atividades desenvolvidas durante cada uma das fases do projeto.

2 Fase de estudo e planejamento

Para que uma nova versão da SmartLock IoT fosse desenvolvida, foi necessário estudar a versão anterior, listar e entender como cada uma das funcionalidades que foram implementadas foi implementada e como se dá a interação entre o dispositivo e o servidor.

2.1 Pesquisa de substitutos para a Raspberry Pi

Foi realizada uma pesquisa de alternativas para a Raspberry Pi [16], hardware utilizado na versão original da solução de controle de acesso. Primeiro, verificou-se a possibilidade de manter o

[1] https://smartcampus.prefeitura.unicamp.br/

uso de um microcomputador por possuir mais memória e mais capacidade de processamento, além de conexão Wi-Fi e entradas e saídas digitais suficientes para os requisitos do projeto. Após pesquisas, não foi encontrada alternativa que contribuísse para reduzir o custo do projeto em relação à versão anterior. Desse modo, o foco da pesquisa por uma placa substituta passou a ser em microcontroladores. Durante a pesquisa os critérios foram preço, suporte da comunidade de desenvolvedores e disponibilidade de bibliotecas, funcionalidades e desempenho computacional. Foi escolhida a placa ESP32 [3] que possui um módulo Wi-Fi e Bluetooth integrado ao microcontrolador e uma implementação da linguagem de programação Python[12] para microcontroladores chamada MicroPython[10], que possui diversas bibliotecas já implementadas, documentação e uma comunidade ativa, o que facilita a obtenção de materiais para aprendizado sobre a ferramenta em questão.

2.2 Estudo de funcionalidades implementáveis

Com a ESP32 [3], iniciou-se uma análise para determinar quais funcionalidades do projeto anterior são implementáveis no novo projeto. Após conversas com os usuários do projeto anterior, um conjunto de funcionalidades mínimas foi estabelecido, sendo essas:

1. o uso do leitor de cartão RFID[13] (*Radio Frequency Identification*), em português, identificação por rádio frequência, que permite armazenar dados em objetos e transmiti-los por meio de rádio frequência. Os cartões/etiquetas possuem *transpoders* que enviam sinais de rádio para uma base transmissora, que no caso do projeto é um leitor; e

2. a abertura por agendamento, que permite que o usuário agende um dia e horário, por meio de um sistema web, para que a fechadura abra automaticamente.

Caso esses requisitos fossem implementados com sucesso, seria estudado a viabilidade de implementação da câmera leitora de QRCode.

2.3 Pesquisa de componentes e bibliotecas

Com os requisitos mínimos estabelecidos iniciou-se uma pesquisa para encontrar os módulos, componentes e bibliotecas que viabilizassem a implementação do projeto. O critério principal foi compatibilidade com a placa. Para o caso de sensores, como o leitor de cartões RFID[13], algum *driver* ou biblioteca para a linguagem MicroPython[10], e por fim preço.

3 Fase de implementação e testes isolados

Um dos desafios seria encontrar um modo de realizar tanto a leitura dos cartões e verificação de autorização para abrir a fechadura quanto verificar se há algum agendamento para o horário vigente, ambos simultaneamente. Um dos motivos para escolha da placa foi a existência de dois núcleos no processador e possibilidade de realizar *multithread*, segundo o manual técnico da Intel [6], é a execução de várias *threads*, em um mesmo processo, o que possibilita realizar múltiplas operações em simultâneo.

3.1 Implementação da leitura do cartão

O passo inicial foi implementar a leitura de um cartão RFID [13]. Para isso, foi escolhido o sensor MFRC-522

3.2 Comunicação com o servidor e configuração de rede

Para realizar a validação de quais cartões podem abrir a fechadura e em quais horários a fechadura devem ficar aberta por agendamento, é necessário que o dispositivo consulte um servidor remoto por meio da Internet usando o protocolo HTTPS [5]. Para isso foi necessário recorrer a uma biblioteca que realiza as requisições HTTPS.

Devido à necessidade de conexão com a Internet também foi preciso configurar a comunicação por Wi-Fi. Para isso, foi utilizada a biblioteca de Wi-Fi nativa da placa. Essas configurações são realizadas no arquivo de *boot*, onde o modulo Wi-Fi é configurado para o modo AP, o qual faz com que o modulo Wi-Fi se comporte como um cliente. Além disso, são fornecidas as credenciais de acesso à rede onde o dispositivo deve ser conectado.

Além das configurações de rede é necessário configurar o relógio interno da placa para que o certificado SSL *(Secure Socket Layer)*[18] seja válido. Para isso, foi utilizado o protocolo NTP *Network Time Protocol* [11] para sincronização do horário da placa com a base nos dados fornecidos pelo servidor da UNICAMP.

3.3 Abordagem para a execução das tarefas

Para que o sistema execute todas as tarefas necessárias para seu funcionamento correto, é necessário que o dispositivo esteja controlando o leitor de cartões e se comunicando com o servidor o tempo todo. Sendo para isso necessário que o código seja escrito de maneira concorrente ou paralela. Programação concorrente, como definido por Dijkstra [15], é o paradigma de programação em que a execução dos processos são alternadas. De modo que esse troca de execução dos processos seja rápida o suficiente para dar a ilusão de que todos são executados simultaneamente. Já na programação paralela, segundo o manual técnico da Intel [6], a ideia é usar os múltiplos núcleos de um processador para executar vários processos em simultâneo.

A opção inicial foi tentar implementar utilizando paralelismo por meio de *multithreading* [6]. Após pesquisas, foi encontrada uma biblioteca, que segundo a documentação oficial [8] encontra-se em estado experimental, que oferecia interface para uso de *multithreading* usando Micropython [10]. Após estudos, foi concluído que não era uma alternativa viável devido à baixa capacidade de armazenamento da placa. O protocolo HTTPS requer um certificado SSL e a placa ESP32 não possui memória suficiente para gerar e armazenar o certificado, e juntamente com outros dados necessários para a execução do programa do modo como foi implementado. Concluiu-se que para as bibliotecas utilizadas em conjunto com o *multithreading* a placa ESP32 não possui capacidade computacional o suficiente para realizar requisições em paralelo utilizando o protocolo HTTPS. Assim sendo, propõem-se outras soluções. Que sao: uso de concorrência[15], de interrupções[7], ou usar o RTOS (*Real Time Operating System*) [14].

3.3.1 Abordagem por uso de RTOS

Um RTOS [14] pode ser descrito como um tipo sistema operacional de proposito geral, uma das principais características diferenciadoras entre sistemas operacionais são os *scheduler*s. Um *scheduler* é a parte do sistema operacional que define quando e por quanto tempo programas terão acesso ao processador do dispositivo em que estão sendo executados. A principal característica do *scheduler* do FREE RTOS [4] é ser previsível, ou seja, ser um *scheduler* determinístico, em que o tempo e prioridade de execução das *threads* seja sempre o mesmo. O RTOS escolhido para o projeto foi o FREE RTOS[4], uma versão compatível com os recursos oferecidos por um sistema embarcado (i.e. memoria RAM e capacidade de processamento). Após estudos e testes, o problema da falta de memória para armazenar o certificado SSL aconteceu novamente. Foi considerado tentar

implementar uma versão simplificada do código que faz as requisições HTTPS e gera o certificado SSL. Dado o tempo e esforço que essa tarefa iria requerer, decidiu-se por outra alternativa que não envolvesse *multithread* e programação paralela.

A Figura 1 apresenta o fluxograma que descreve o fluxo de execução para a abordagem que faz uso de programação paralela e *multithreading*.

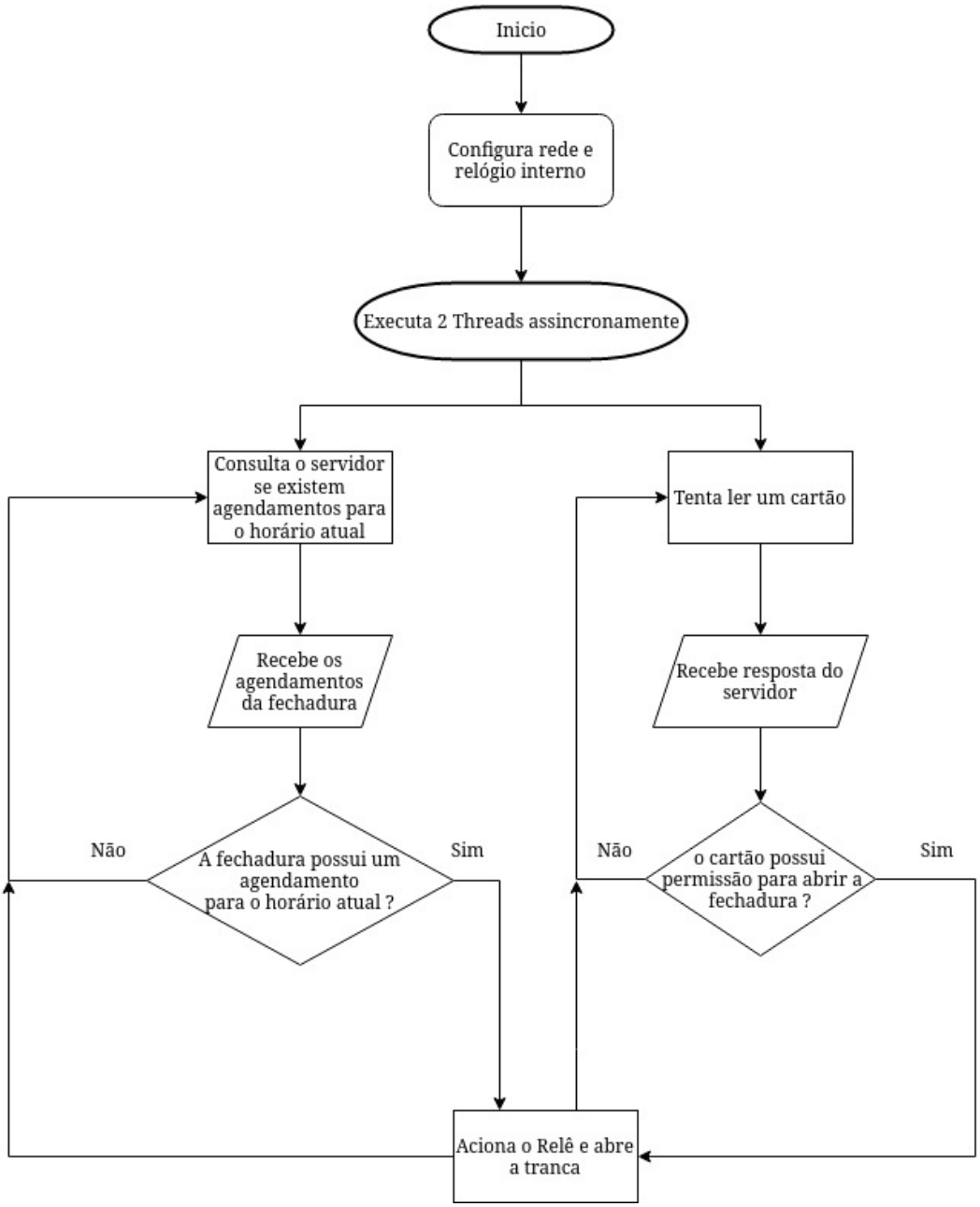

Figura 1: Versão com paralelismo [Fonte: Autor]

3.3.2 Uso de Interrupções

Durante uma leitura do *datasheet* do módulo leitor de cartões, foi identificado um recurso de interrupção como definido no artigo [7]. Com este recurso, quando um sinal ou variável booleana muda de valor, a execução do processo atual é interrompida e uma sub-rotina é executada. Após o fim da execução dessa rotina, o processo retorna a execução do ponto onde foi interrompido. Segundo o datasheet [9], quando um cartão é detectado pelo módulo, um sinal é enviado no pino de interrupção. A existência dessa função abre a possibilidade de apenas executar a tarefa de ler e verificar a permissão de acesso de um cartão quando o leitor detectasse um cartão. Tal função permite que enquanto esse sinal não fosse enviado pelo módulo, a placa ESP32 executasse a rotina de verificação de agendamento de abertura. A Figura 2 apresenta o fluxograma que descreve o fluxo de execução para a abordagem que faz uso de interrupções.

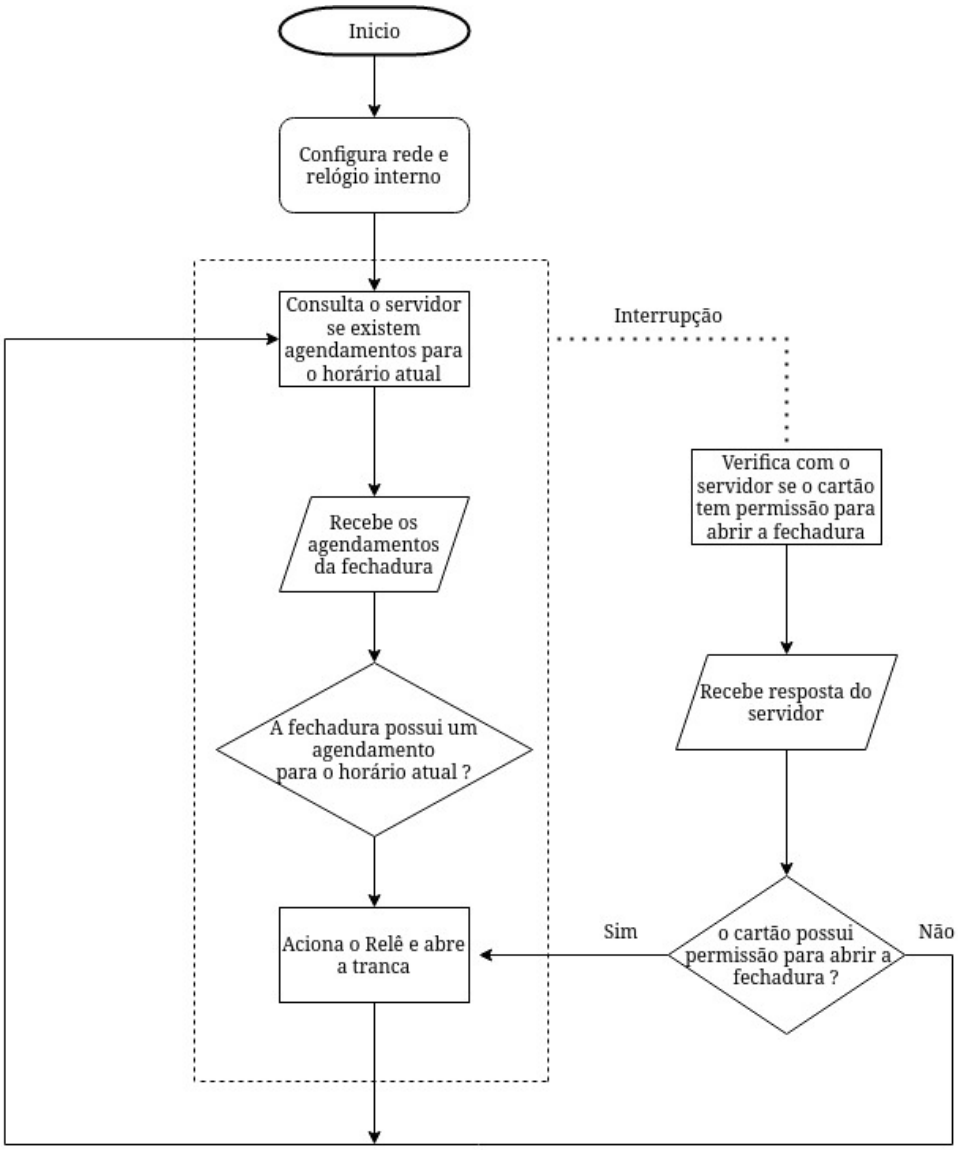

Figura 2: Fluxograma da versão com interrupções [Fonte: Autor]

Porém, o *driver* utilizado não implementou suporte para a função de interrupção. Foi considerado modificar o *driver* para que esse problema fosse resolvido, mas a falta de documentação e complexidade da tarefa fez com que essa possibilidade fosse descartada.

3.3.3 Assincronismo por meio de concorrência

Após pesquisas foi encontrada uma biblioteca [1] que implementa assincronismo para Micropython. A partir dessa biblioteca, desenvolveu-se uma versão do programa que funciona por meio de concorrência, onde em momentos de ociosidade do programa o uso do processador é cedido para a outra tarefa e alterna entre tarefas durante a execução. A Figura 3 apresentada o fluxograma para a implementação que usa concorrência.

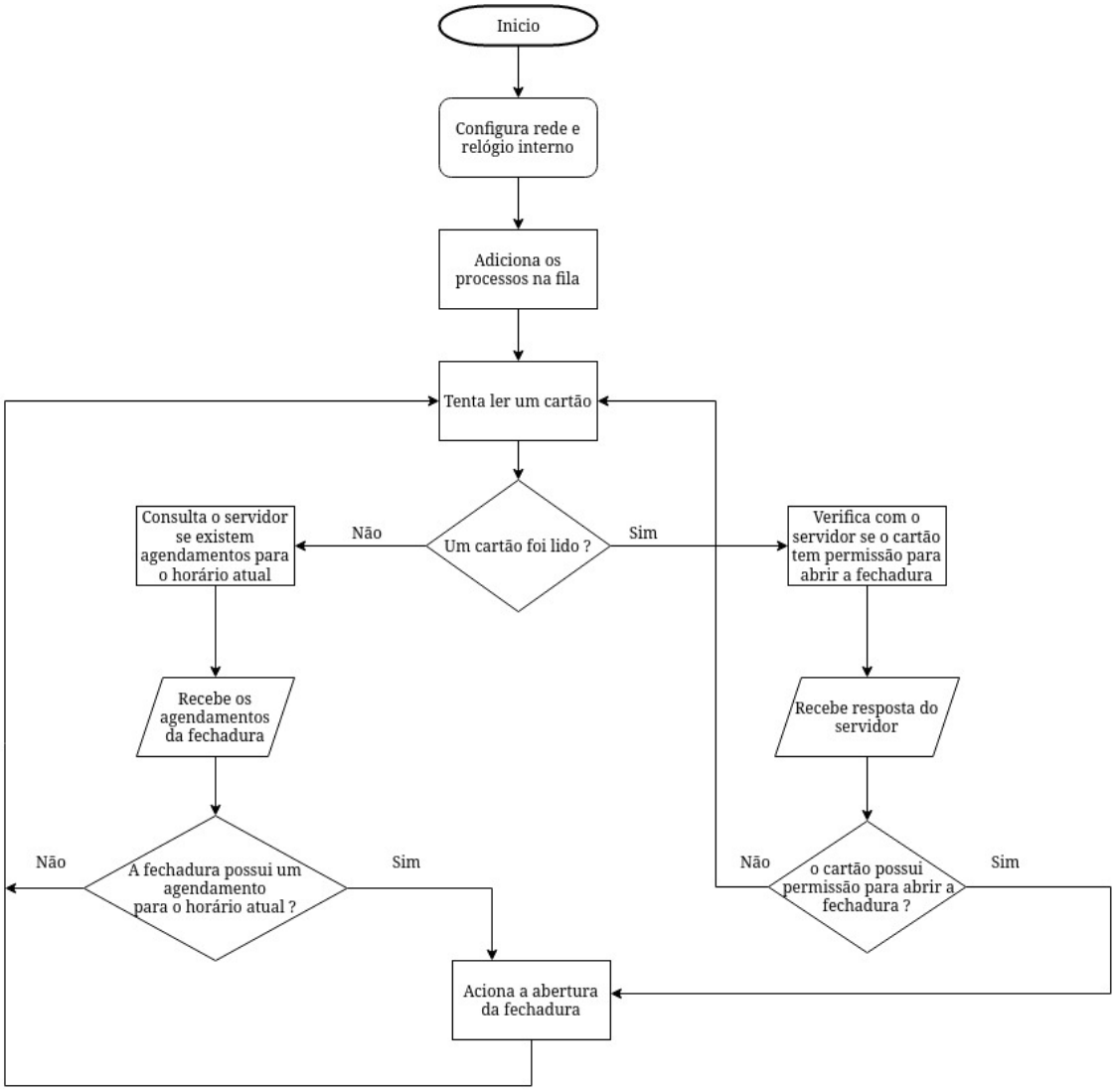

Figura 3: Fluxograma da versão assíncrona [Fonte: Autor].

4 Fase de Testes

Após concluir o desenvolvimento do programa de maneira separada foi iniciada a integração dos módulos de software e hardware (i.e. leitor de cartões, LEDs de identificação, sistema de agendamento). Inicialmente, os componentes foram conectados por meio de uma protoboard para teste. A Figura 4 apresenta o diagrama de conexões e a Figura 5 apresenta o circuito em uma *protoboard*.

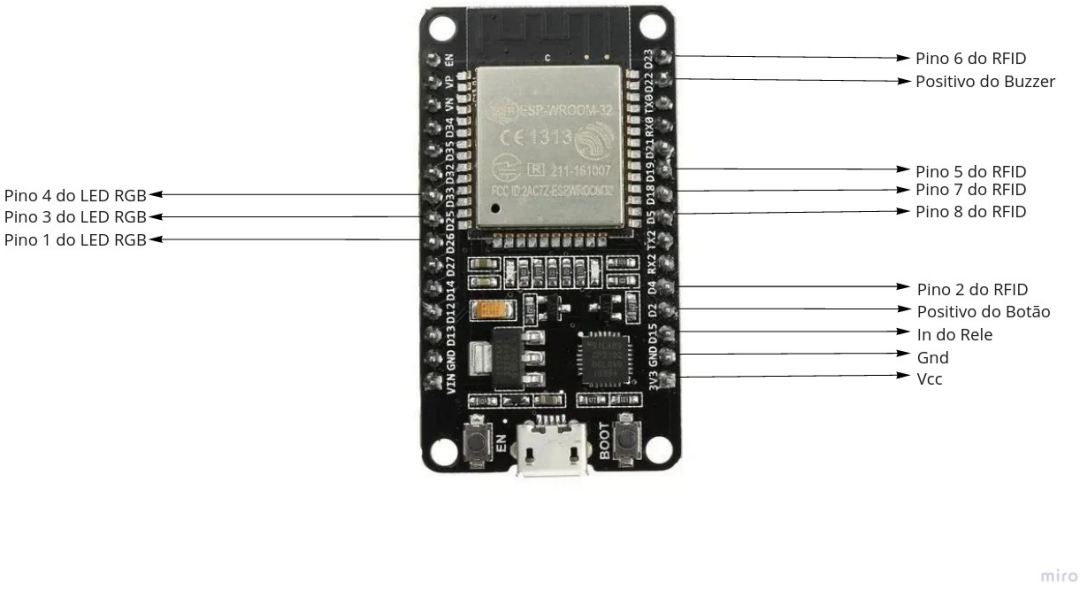

Figura 4: Diagrama de conexões

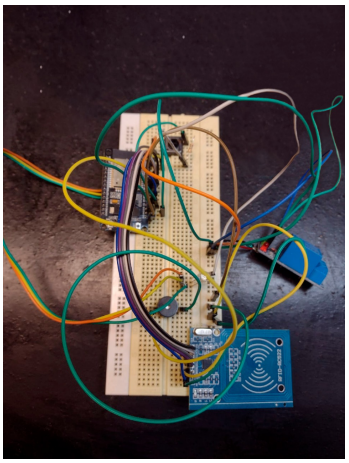

Figura 5: Circuito na protoboard [Fonte: Autor]

Para verificar o funcionamento correto do protótipo, este foi mantido em funcionamento por 14 dias simulando seu uso em ambiente real. Após verificar que o funcionamento estava correto, foi iniciado o desenvolvimento de uma versão em placa de cobre perfurada, que pode ser vista na Figura 6.

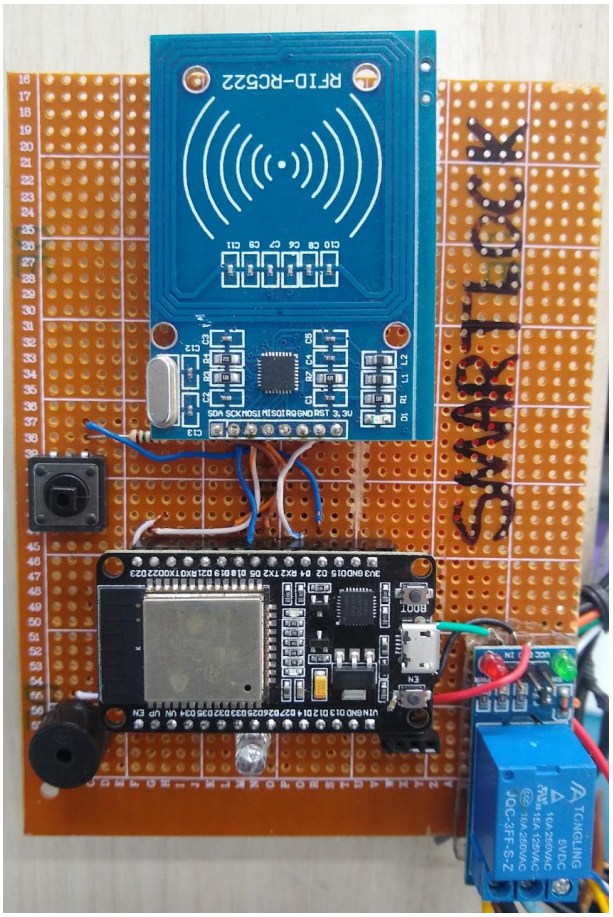

Figura 6: Circuito em Placa Perfurada [Fonte: autor]

O novo processo de testes durou 7 dias. Após os testes, foi construída uma maquete para apresentação do projeto, que pode ser vista na Figura 7.

Figura 7: Maquete de apresentação [fonte: Autor]

4.1 Lista dos componentes de hardware utilizados

Os componentes utilizados na versão final do projeto estão listados a seguir:
1 Módulo ESP32
1 Módulo MFRC522
1 Módulo Rele
1 Buzzer
1 Led RGB
1 Chave Tactíl
1 Resistor de 150 Ohms 10

5 Conclusão

O uso da ESP32[3] apresentou-se viável tendo em mente os requisitos listados anteriormente, os quais são: leitura e validação de Cartões RFID [13] e agendamento de abertura da fechadura. A implementação das interfaces e *driver*s para MicroPython ainda são recentes e experimentais [8]. Há falta de suporte para *multithread* [6] e o uso de mais funcionalidades da biblioteca *asyncio*[1] para um melhor proveito da ideia de concorrência. O uso de RTOS, por outro lado, é mais promissor, pois oferece controle sobre o uso do processador e recursos da placa, mas requer mais estudos para fazer proveito de todas as funcionalidades.

Para uma próxima implementação desse projeto, faz-se necessário procurar por um microcontrolador ou microcomputador que possua maior desempenho computacional. Melhorias que podem ser implementadas futuramente envolvem implementar assincronismo na biblioteca de requisições HTTPS [5], implementar uma interface para *multithreading*, implementar interrupções para o leitor de cartões RFID e projetar uma versão do circuito em PCI (Placa de Circuito Impresso), para uma versão mais compacta.

Agradecimentos

Gostaria de agradecer a Equipe do SmartCampus Unicamp por toda a orientção e suporte durante a realização deste trabalho.

Referências

[1] “Uasyncio - Asynchronous I/O scheduler¶,” uasyncio - asynchronous I/O scheduler - MicroPython latest documentation, 07-Dec-2022. [Online]. Available: https://docs.micropython.org/en/latest/library/uasyncio.html. [Accessed: 07-Dec-2022].

[2] Dijkstra, E. Co-operating sequential processes. (Academic Press Inc.,1968)

[3] “ESP32,” ESP32 Wi-Fi amp; Bluetooth MCU I Espressif Systems. [Online]. Available: https://www.espressif.com/en/products/socs/esp32. [Accessed: 03-Dec-2022].

[4] “Market leading RTOS (real time operating system) for embedded systems with internet of things extensions,” FreeRTOS, 10-Oct-2022. [Online]. Available: https://www.freertos.org/. [Accessed: 12-Dec-2022].

[5] “HTTPS - glossário do MDN web docs: Definições de Termos relacionados à web: MDN,” Glossário do MDN Web Docs: Definições de termos relacionados à Web — MDN. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Glossary/https. [Accessed: 03-Dec-2022].

[6] Here’s how Intel® Hyper-Threading Technology (Intel® HT Technology) helps processors do more work at the same time.1 Here’s how Intel® Hyper-Threading Technology (Intel® HT Technology) hel, “What is hyper-threading?,” Intel. [Online]. Available: https://www.intel.com.br/content/www/br/pt/gaming/resources/hyper-threading.html. [Accessed: 06-Dec-2022].

[7] Zelkowitz, M. Interrupt Driven Programming. *Commun. ACM.* **14**, 417-418 (1971,6), https://doi.org/10.1145/362604.362618

[8] “_thread – *multithreading*[6] support¶,” _thread– *multithreading*[6] support - MicroPython latest documentation, 06-Dec-2022. [Online]. Available: https://docs.micropython.org/en/latest/library/_thread.html. [Accessed: 06-Dec-2022].

[9] Alldatasheet.com, “MFRC522 datasheet(pdf) - NXP semiconductors,” ALLDATASHEET.COM - Electronic Parts Datasheet Search. [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/227839/NXP/MFRC522.html. [Accessed: 12-Dec-2022].

[10] “MicroPython,” Python for microcontrollers. [Online]. Available: https://micropython.org/. [Accessed: 03-Dec-2022].

[11] “O NTP” NTP.br. [Online]. Available: https://ntp.br/conteudo/ntp/. [Accessed: 06-Dec-2022].

[12] “Welcome to Python.org,” Python.org. [Online]. Available: https://www.python.org/about/. [Accessed: 03-Dec-2022].

[13] R. Want, ”An introduction to RFID technology,”in IEEE Pervasive Computing, vol. 5, no. 1, pp. 25-33, Jan.-March 2006, doi: 10.1109/MPRV.2006.2.

[14] “Why RTOS and what is RTOS?,” FreeRTOS, 27-Nov-2019. [Online]. Available: https://www.freertos.org/about-RTOS.html. [Accessed: 03-Dec-2022].

[15] Dijkstra, E. Co-operating sequential processes. (Academic Press Inc.,1968)

[16] Raspberry Pi, "Buy A raspberry pi 3 model B," Raspberry Pi. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/. [Accessed: 03-Dec-2022].

[17] S. C.- Unicamp, "Smart campus," Unicamp. [Online]. Available: https://smartcampus.prefeitura.unicamp.br/. [Accessed: 12-Dec-2022].

[18] "Secure sockets layer (SSL) - MDN web docs glossary: definitions of web-related terms: MDN," MDN Web Docs Glossary: Definitions of Web-related terms — MDN. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/SSL. [Accessed: 03-Dec-2022].

[19] Unicamp. [Online]. Available: https://www.unicamp.br/unicamp/. [Accessed: 12-Dec-2022].