UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DAINF - DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

GUSTAVO LUIZ ANDRADE CORRÊA

SISTEMA OPEN-SOURCE PARA A ATUALIZAÇÃO DE FIRMWARE OTA(OVER-THE-AIR) BASEADO NAS BIBLIOTECAS LWIP, MBEDTSL E FATFS

PROPOSTA DE TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2019

GUSTAVO LUIZ ANDRADE CORRÊA

SISTEMA OPEN-SOURCE PARA A ATUALIZAÇÃO DE FIRMWARE OTA(OVER-THE-AIR) BASEADO NAS BIBLIOTECAS LWIP, MBEDTSL E FATFS

Proposta de Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de engenheiro de computação.

Orientador: Prof. Dr. Gustavo Weber Denardin
Departamento Acadêmico De Elétrica

PATO BRANCO
2019

SUMÁRIO

1 INTRODUÇÃO

Com a evolução da microeletrônica e, por consequência, a redução de custo de periféricos e o crescimento do poder computacional de processadores, os sistemas computacionais se tornaram cada vez mais pequenos e baratos. Devido a isso, processadores e microcontroladores passaram a ser instalados em produtos, o que deu origem ao conceito de sistema embarcado, que são sistemas de processamento de informação embutidos em produtos (MARWEDEL, 2006). A utilização desses sistemas foi disseminada em várias áreas como, a automobilística, aeronáutica, ferroviária, industrial, médica, entre outras, automatizando as mais diversas funções. Em algumas dessas funções era fundamental a presença de agentes humanos para serem realizadas, ou não existiam, pois uma pessoa não a exerceria em tempo hábil.

Os sistemas computacionais embarcados são compostos pelos mesmos componentes utilizados para a constituição de computadores pessoais, porém com tamanhos, capacidades e custos reduzidos. Tais dispositivos operam de forma independente e geralmente são projetados para realizar tarefas específicas e repetitivas. Sistemas embarcados estão presentes no dia a dia da maioria das pessoas, em micro-ondas, geladeiras, TVs, aparelhos de som, video games e outros produtos eletrônicos (MARWEDEL, 2006), logo, esses dispositivos se distanciam dos computadores de propósito geral, como vemos em *desktops* e *notebooks* atuais.

Com a necessidade cada vez maior da implementação desses sistemas no nosso dia a dia, é imprescindível se obter *hardwares* e *softwares*, cada vez mais robustos e que atendem todas as necessidades dos seus usuários. Assim, o projeto desses produtos devem ser muito bem planejado, e executado de forma a serem entregues produtos de qualidade, à prova de falhas e que possam reagir a erros, de forma a não causar danos a seus utilizadores.

Durante a fase de projeto de um sistema embarcado, deve-se avaliar diversos âmbitos, como desempenho, confiabilidade, consumo de energia, manufaturabilidade etc. É também necessário validar essas avaliações, com o intuito de verificar se atenderão os requisitos de projeto, e pela necessidade desses produtos serem eficientes, é indispensável que esses sistemas passem por uma fase de otimização, em que mudanças no projeto podem melhorar a eficiência energética do produto ou até mesmo gerar novas funcionalidades a esses equipamentos. Portanto o projeto como um todo precisa ser testado para evitar que erros e *bugs* possam vir a permanecer no produto final (MARWEDEL, 2006), criando um ciclo de desenvolvimento que deve ser repetido até se obter um produto eficiente, de qualidade e completo.

Após a instalação final desse projeto para seu cliente, eventualmente pode ser necessária uma nova funcionalidade, uma otimização ou então, podem ser exigidos testes nesse sistema. Logo, é preciso que haja uma forma de se alterar esse produto mesmo após seu lançamento, para assim gerarmos um maior valor e confiabilidade ao sistema. A possibilidade de serem feitas manutenções futuras no *software*, que no contexto de sistemas embarcados é chamado de *firmware*, conhecida como atualização OTA (*Over-The-Air*). Esse recurso não é obrigatório

no projeto de um sistema embarcado, mas é muitas vezes necessário, podendo ser uma funcionalidade muito útil dependendo da aplicação do sistema em concepção. A decisão de utilizar ou não a atualização OTA pode influenciar na escolha do *hardware* utilizado no projeto (BALL, 2002), podendo aumentar o custo do produto final. Uma das principais soluções adotadas para a manutenção desses programas é criar métodos de atualização em que, é necessária a presença de um agente humano fisicamente próximo do sistema para fazer a manutenção do *software*, o que acaba aumentando o custo de manutenção do produto e o tornando menos atrativo para os seus compradores.

Este trabalho de conclusão de curso propõem um método de manutenção desses *firmwares* de forma remota, que possa ser o mais portável possível. Na solução proposta, o dispositivo embarcado poderá verificar periodicamente um servidor a procura de uma nova versão do seu *firmware*. Quando encontrado, será realizado o *download* do novo *software* para a memória interna do dispositivo, para posterior atualização do equipamento. O diferencial da abordagem proposta é basear a solução em bibliotecas amplamente difundidas em sistemas embarcados, como LwIP (DUNKELS, 2002), Mbedtls (DEVINE, 2006) e FatFS(CHAN, 2016). Dessa forma, o código do sistema de atualização é totalmente portável, desde que a plataforma escolhida tenha suporte a tais bibliotecas. A única peça de *software* que não será totalmente portável será o *bootloader* que substituirá o *firmware* antigo pelo novo na memória *flash* do dispositivo, por ser dependente do *hardware* utilizado.

Os *bootloaders* estão atualmente presentes em todos os computadores pessoais e em alguns sistemas embarcados. Esse *software* prepara a maioria dos *hardwares* presentes na máquina para um sistema operacional ou outro programa entrar em ação. Como é o primeiro programa a ser inicializado após um sistema ser iniciado ou após um *reset*, ele pode ter várias funções, como, realizar checagem de periféricos, verificar se o *firmware* presente na memória não está corrompido, além de poder fazer a troca do *software* presente na memória (DAVIS; DURLIN, 2013), que será sua principal utilização neste trabalho.

Um dos seus principais usos é em *smartphones*, em que são utilizados para a atualização de sistemas operacionais como *Android* e iOS, e como garantia de restauração em caso de erros irreversíveis no sistema operacional. É desenvolvido pelo próprio fabricante do dispositivo, e por padrão é bloqueado para os usuários, evitando a substituição do *software* original do aparelho por uma versão customizada, mas ainda assim existem opções de desbloqueio do *bootloader*, dependendo do modelo do aparelho e do fabricante (SALUTES, 2018).

1.1 OBJETIVO GERAL

Este trabalho de conclusão de curso tem como objetivo geral o desenvolvimento de um sistema *open-source* para a atualização de *firmwares* OTA*(Over-The-Air)* de sistemas embarcados baseado nas bibliotecas FatFs, LwIP e mbedTLS.

1.2 OBJETIVOS ESPECÍFICOS

- Desenvolver o *bootloader* que identifica versões e atualiza o *firmware*.
- Criar um servidor HTTP para a comunicação cliente-servidor.
- Implementar a tarefa que fará a comunicação segura entre o servidor HTTP e a plataforma embarcada, verificará de disponibilidade de atualização e fará o *download* da nova versão, se existente, será baseando nas bibliotecas LwIP e mbedTLS.
- Produzir a tarefa de leitura de um cartão micro SD que guarda uma nova versão do *firmware* e uma cópia do anterior, utilizando a biblioteca FatFs.
- Comprovar o funcionamento da técnica de atualização remota de *firmware*, utilizando a plataforma embarcada STM32F7.

2 REVISÃO DE LITERATURA

2.1 LINGUAGEM DE PROGRAMAÇÃO C

2.2 PLATAFORMA EMBARCADA — STM32F7 DISCOVERY

2.2.1 ARM M7 E SUA FAMÍLIA.

2.3 SERVIDORES HTTP

2.4 BOOTLOADER

2.4.1 LINKER

2.5 LWIP

2.6 MBEDTLS

2.7 FATFS

3 METODOLOGIA

Com a utilização do *kit* de desenvolvimento STM32F7 *Discovery*, irão ser programadas as tarefas e o *bootloader* que serão os principais componentes desse sistema. As tarefas serão produzidas a partir de bibliotecas já conhecidas e vastamente utilizadas por desenvolvedores de sistemas embarcados, para que assim projetos que necessitem fazer comunicação segura via rede e leitura e escrita de cartões SD, possam utilizar esse sistema de modo a poupar espaço na memória, visando a reutilização dessas bibliotecas, portanto o sistema pode ser amplamente utilizado.

A partir de um arquivo de *linker*, a memória da plataforma será customizada a fim de abrigar os arquivos necessários para o sistema e protege-los de eventuais sobre-escritas que podem vir a ocorrer. Esse arquivo de *linker*, assim como o *bootloader*, será escrito somente para a plataforma STM32F7, visto que cada plataforma tem suas próprias características como, tamanho de memória e endereços diferentes para cada fabricante e/ou arquitetura. A seguir será explicado parcialmente como funcionarão as funções das tarefas, do *bootloader*, e do servidor HTTP.

3.1 O *BOOTLOADER*

O *bootloader* será responsável em fazer a validação e troca de cada versão de *firmware* instalado no sistema embarcado. Sempre que o sistema for iniciado, o *bootloader* fará a procura de um novo *firmware* na memória interna do sistema. Esse processo, irá verificar o *hash* da nova versão, verificando a integridade e origem do *software*, para então poder ocorrer a atualização. Caso haja alguma falha durante esse processo, o *bootloader* terá a habilidade de verificar esse erro e corrigi-lo, revertendo a atualização, e instalando o *firmware* anterior. O funcionamento do *bootloader* pode ser observado na figura 1.

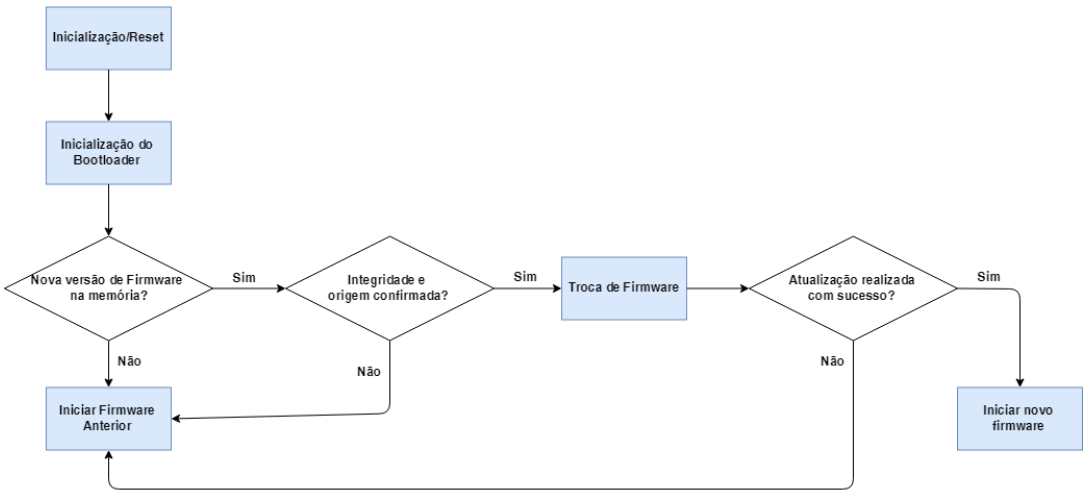

Figura 1 – Diagrama de funcionamento do *bootloader*. Fonte: autoria própria.

3.2 SERVIDOR HTTP

A partir de um computador conectado à mesma rede que o sistema embarcado, haverá um servidor HTTP que ficará responsável por esperar requisições do dispositivo, para consultar a disponibilidade de uma versão atualizada do *software*, e após a confirmação dessa nova versão, esse servidor irá enviar o *firmware* para a plataforma embarcada.

3.3 TAREFAS DO SISTEMA

3.3.1 COMUNICAÇÃO

Com o uso da biblioteca LwIP e MbedTLS essa tarefa estará responsável por criar uma comunicação segura entre o *hardware* e o servidor HTTP, a partir dessa comunicação será feito a verificação do sinal de disponibilidade de novo *software* e *download* do mesmo quando o sistema estiver ocioso.

3.3.2 ARMAZENAMENTO

A partir da utilização da biblioteca FatFs, essa tarefa fará a leitura e escrita sobre um cartão SD instalado no *hardware*. Quando houver a transferência de uma nova versão, tanto o programa anterior quanto o novo, serão colocadas nesta memória, para futuras instalações que serão realizadas pelo *bootloader*. A escolha da mídia de armazenamento de versões do *software* estará a cargo do projetista, sendo escolhido para esse trabalho um cartão SD.

4 CRONOGRAMA

1. Estudo bibliográfico;
2. Desenvolvimento da proposta;
3. Familiarização com plataforma;
4. Projeto de *bootloader* e tarefas;
5. Entrega do TCC1;
6. Desenvolvimento do *bootloader*;
7. Criação do servidor HTTP;
8. Implementação das tarefas;
9. Testes de integração;
10. Escrita da monografia e artigo científico.

Atividade	**08/19**	**09/19**	**10/19**	**11/19**	**12/19**	**01/20**	**02/20**	**03/20**	**04/20**	**05/20**
1	X	X	X	X	X	X	X	X	X	X
2	X									
3	X	X	X	X						
4		X	X	X	X					
5					X					
6					X	X	X			
7						X				
8						X	X	X	X	
9								X	X	X
10	X	X	X	X	X	X	X	X	X	X

Referências

BALL, S. **Embedded Microprocessor Systems: Real World Design**. 3rd. ed. Newton, MA, USA: Butterworth-Heinemann, 2002. ISBN 0750675349. Citado na página 2.

CHAN. **FatFs - Generic FAT File System Module**. 2016. Disponível em: <(http://elm-chan.org/fsw/ff/00index_e.html)>. Acesso em: 06 de setembro de 2019. Citado na página 2.

DAVIS, T.; DURLIN, D. **Bootloaders 101: making your embedded design future proof**. 2013. Disponível em: <(https://www.embedded.com/design/prototyping-and-development/4410233/Bootloaders-101--making-your-embedded-design-future-proof)>. Acesso em: 06 de setembro de 2019. Citado na página 2.

DEVINE, C. **SSL Library mbed TLS / PolarSSL**. 2006. Disponível em: <(https://tls.mbed.org)>. Acesso em: 06 de setembro de 2019. Citado na página 2.

DUNKELS, A. **lwIP - A Lightweight TCP/IP stack**. 2002. Disponível em: <(https://savannah.nongnu.org/projects/lwip/)>. Acesso em: 06 de setembro de 2019. Citado na página 2.

MARWEDEL, P. **Embedded System Design**. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 1402076908. Citado na página 1.

SALUTES, B. **Bootloader: o que é e para que serve?** 2018. Disponível em: <(https://www.androidpit.com.br/bootloader-o-que-e-para-que-serve)>. Acesso em: 06 de setembro de 2019. Citado na página 2.