Uso de servidor web *Node.js* para o controle remoto de portas GPIO do Raspberry Pi

Vitor Bruno de Oliveira Barth¹, ², Pedro Felipe Gonçalves de Arruda¹, ², Ruy de Oliveira¹, ²

1Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso – Campus Cuiabá “Octayde Jorge da Silva”

2Grupo de Pesquisa em Redes e Segurança do IFMT – Campus Cuiabá

vitor.barth@gmail.com

pedro\_felipe\_ga@gmail.com

ruy.oliveira@cba.ifmt.edu.br

***Resumo:*** *Devido ao seu tamanho, hardware e preço, o Raspberry Pi é uma das principais escolhas para automação e monitoramento remoto. Contudo, para ser acessado via web, é necessária a utilização de uma linguagem intermediária como C ou Python para ligar o Servidor Web ao Hardware. Buscando um modo mais simples de controlar remotamente o Raspberry Pi foi testado o servidor web Node.js que possui extensões que controlam os periféricos conectados às portas GPIO, utilizando somente JavaScript. Node.js mostrou-se uma boa escolha devido a fácil implementação, porém algumas funcionalidades são limitadas pelo hardware do Raspberry Pi.*

***Palavras-Chave:*** *Node.js, Raspberry Pi, monitoramento remoto*

1. Introdução

O baixo preço, combinado a um *hardware* eficiente, versátil e completo fazem do *Raspberry* *Pi* um componente útil em soluções de automação (MAKSIMOVIĆ *et al.*, 2014) e experimentação remota (CROTTI *et al.*, 2013). Uma de suas principais características é o conjunto de pinos (26 pinos no modelo A, e 40 pinos nos modelos A+, B e B+) para entrada e saída de dados, as quais servem como interface física entre o servidor e o restante do mundo. Por fornecer suporte a uma grande quantidade de periféricos e à comunicação em rede o *Raspberry* *Pi* é utilizado principalmente como sensor e controlador remoto (VUJOVIĆ & MAKSIMOVIĆ, 2014).

A fim de que o *Raspberry* *Pi* se comunique em rede, é necessária a instalação de um servidor *web*, o qual tem por objetivo enviar comandos aos periféricos de saída ou ler e apresentar informações de periféricos de entrada partindo de uma requisição remota via *web*.

Para que periféricos devolvam informações ou executem ações por pedido do servidor *web*, ou seja, para que o *Raspberry* *Pi* realize alguma tarefa de escrita ou leitura a nível de *hardware* partindo de uma chamada de *software,* costuma-se executar programas escritos em linguagens de programação *C* ou *Python* por meio de *scripts* em *PHP (PHP: Hypertext Preprocessor).* (CROTTI *et al.*, 2013; MAKSIMOVIĆ *et al.*; 2014, VUJOVIĆ & MAKSIMOVIĆ, 2014).

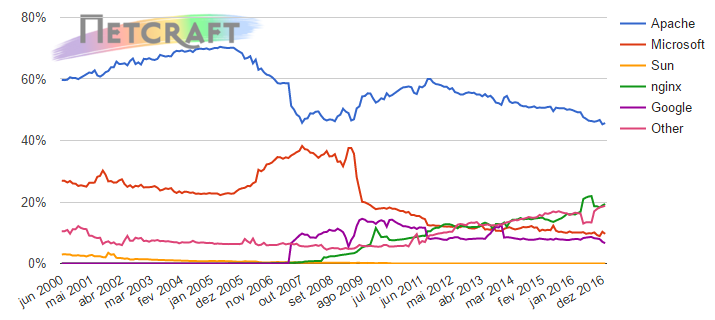
O mercado de servidores *web* é hoje dominado pelo *Apache*, que sozinho representa 50% do mercado de servidores, em seguida aparecem *nginx,* *Microsft IIS* e *Google Web Server* que ao todo administram 85% dos sites ativos, de acordo com pesquisas realizadas pela empresa de assessoria e segurança virtual *Netcraft,* conforme apresentado na figura 1*.*

Figura 1 - Mercado de servidores web de sites ativos. Fonte: Netcraft (Acesso em 02/03/2016)

Por mais que sejam populares, nenhum destes servidores consegue controlar as portas de entrada e saída de dados do *Raspberry Pi,* necessitando de uma ferramenta externa que execute códigos em linguagem *C* ou *Python*.

*Node*.*js* (ou simplesmente *Node*) é um servidor *web* implementado em *C* e C*++,* desenvolvido em 2009 por Ryan Dahl, utilizando o motor *JavaScript* V8 produzido pelo *Google*. Devido à facilidade de desenvolvimento de extensões*,* muitos pacotes auxiliares foram produzidos para que o *Node.js* realizasse as mais diversas funções, dentre elas, controlar portas GPIO nativamente.

Experimentos conduzidos por McCune (2011) e Ribeiro Jr. (2012) mostraram que *Node*.*js* possui performance superior aos seus concorrentes, administrando maior número de requisições em baixo tempo, o que torna viável a implementação de servidores web *Node.js* em larga escala.

O objetivo deste artigo é apresentar um modo de controlar o *hardware* e as requisições *web* utilizando somente *JavaScript* por meio de um servidor Node*.*js, evitando que seja necessária uma linguagem intermediária para executar comando em baixo nível.

Este artigo está estruturado em 4 seções, incluindo a introdução. A seção 2 descreve os materiais, métodos e configurações utilizados para se implementar um servidor *Node.js* utilizando *Raspberry* *Pi*. A seção 3 explica o uso do *Node.js* para tarefas de escrita e leitura analógicas e digitais. Na seção 4 são apresentados os resultados, dificuldades e conclusões obtidas durante os experimentos apresentados neste artigo.

2. Materiais e Métodos

A fim de comprovar que um servidor *Node* pode realizar tarefas de escrita e leitura de dados digitais e analógicos realizou-se experimentos usando periféricos que enviam ou recebem estes tipos de informação.

Como exemplo de entrada de dados buscou-se interpretar, em *software*, os dados digitais enviados por um botão *push* e os dados analógicos enviados por um potenciômetro de 22 kΩ.

Para verificar a saída de dados digitais e analógico visa-se criar interruptores para três *LEDs* difusos e um controle de volume para um sonorizador *piezo*, também conhecido como *buzzer.*

Estes quatro periféricos foram conectados a um *Raspberry* *Pi* 3 Modelo B utilizando uma placa de prototipagem com 640 pontos e um adaptador de GPIO para a *protoboard.* Como a placa Raspberry Pi 3 possui somente entradas e saída de dados digitais, fez-se necessário o uso do circuito integrado MCP3008, um conversor Analógico-Digital com interface SPI de 8 *Bits*.

No *Raspberry* *Pi* 3 está instalado o sistema operacional *Raspbian* *Jessie* com *Pixel* de 2017-03-02, utilizando *kernel* *Linux* 4.4.5. O servidor *Node*.*js* foi instalado em sua versão 6.4.0 LTS, distribuída pelo sistema operacional através da ferramenta *APT* (*Advanced Packaging Tool)*.

O controle das portas GPIO do Raspberry Pi 3 por meio do servidor *Node*.*js* faz uso da biblioteca *node-rpio* na versão 0.9.15. Esta biblioteca foi desenvolvida e mantida por Jonathan Perkin e está disponível por meio do *GitHub*. Para facilitar a administração de requisições *HTTP* e transferência de arquivosforam utilizadas as bibliotecas *express* v4.14.1 e *cors* v2.8.1. O download e instalação destas bibliotecas foi realizado através da versão 3.10.10 da ferramenta NPM (*Node Package Manager)*.

Foi desenvolvida uma página básica em *HTML5* e *CSS3*, utilizando o tema *Astral*, distribuído gratuitamente sob licença *Creative* *Commons* pelo site *HTML5* *Up*!, e os *frameworks* *jQuery* v1.11.3 e *Bootstrap* v3.3.7, testada nos navegadores para *Google* *Chrome* 55.0.2883 e *Mozila* *Firefox* 51.0.1.

3. Aplicação piloto

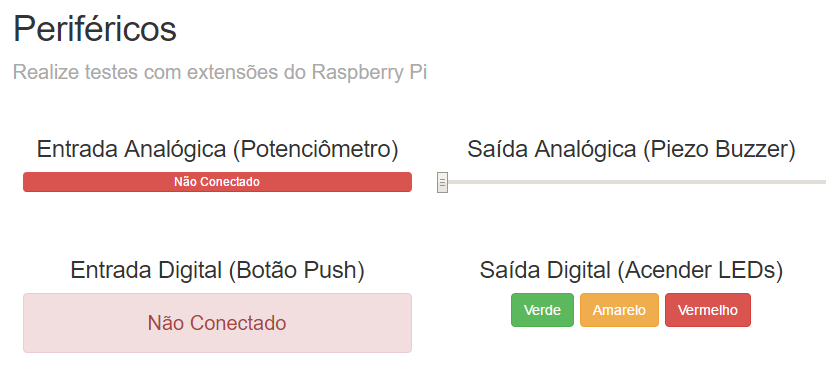
A aplicação proposta por este trabalho é tornar a página *web*, apresentada na Figura 2,funcional utilizando apenas um servidor *Node*.*js* através da biblioteca *node-rpio.* Esta página não necessita estar salva no *Raspberry* *Pi*, nem na mesma rede local, desde que possa enviar requisições *HTTP* para o servidor *Node*.*js* sendo executado no *Raspberry* *Pi*.

Figura 2 - Aplicação Piloto (Desenvolvida pelo Autor)

Para que a página se comunique em tempo real com o servidor, as informações serão enviadas e recebidas por meio das requisições *AJAX*. Para a escrita, a página envia ao servidor a informação da ação a ser tomada, como é mostrada na Figura 3.

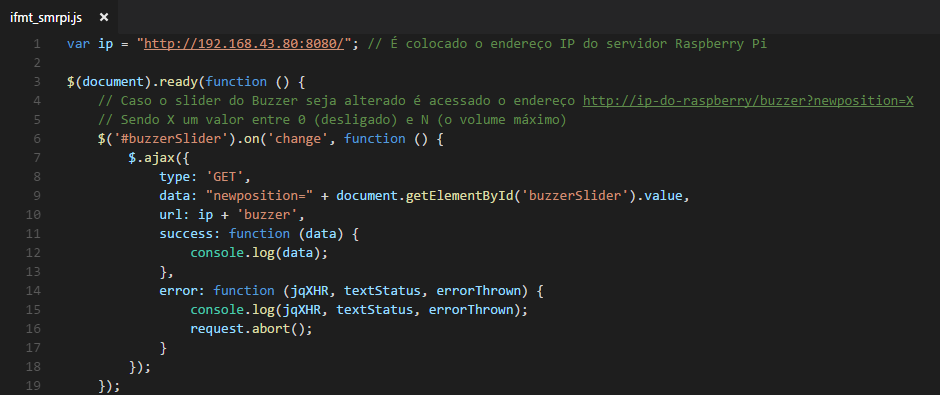


Figura 3 - Requisição AJAX para escrita. (Desenvolvida pelo Autor).

No caso de requisições para LEDs usa-se o formato *http://ip-do-raspberry/LED?color=X,* sendo *X* a cor do LED qual será acendido ou apagado.

Para a escrita, uma requisição *AJAX* é executada sequencialmente em um intervalo predefinido. Ao executar esta requisição, o servidor deverá retornar um valor que representa a informação que a página está buscando, como mostra a Figura 4, e a partir desta informação a página é atualizada.

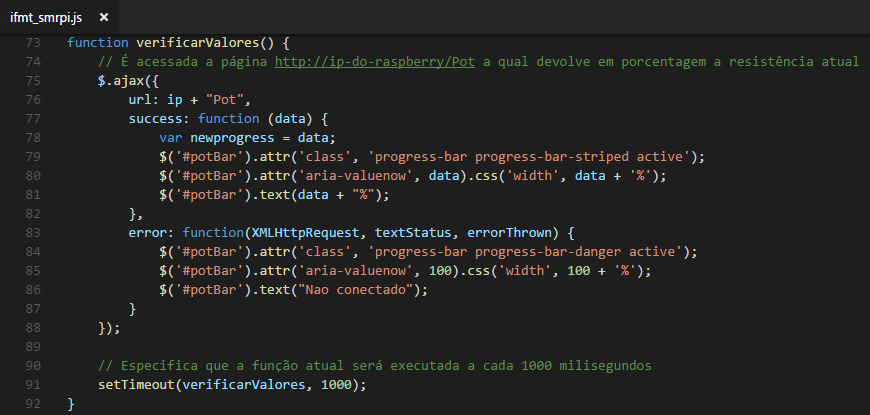


Figura 4 - Requisição AJAX de leitura. (Desenvolvida pelo Autor)

Para a leitura da posição do botão, é acessada a página no formato *http://ip-do-raspberry/Button*, a qual devolve um valor Booleano (*True* ou *False)* que corresponde ao estado atual dele: apertado ou não.

**3.1 Servidor *Node.js* no *Raspberry Pi***

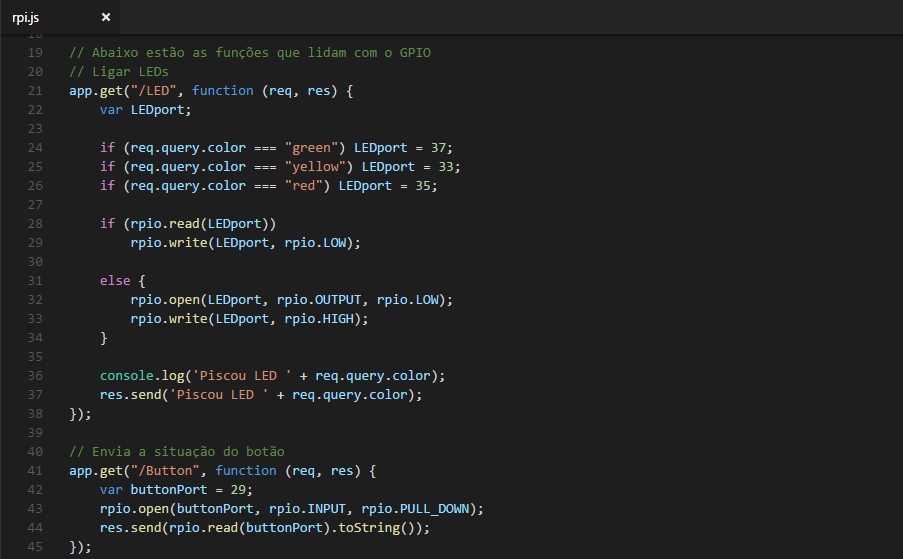
O servidor que está recebendo as solicitações da página web é dividido em quatro funções principais, sendo cada uma delas responsável por um dos periféricos. As requisições das páginas ‘/*LED’* e ‘/*Button’* tratam, respectivamente, de escrita e leitura digitais, enquanto as requisições de ‘/*Buzzer’* e ‘/*Pot’* tratam de escrita e leitura analógica, como apresentado na Figura 5.

Figura 5 - Funções de escrita e leitura analógicas. (Desenvolvida pelo Autor)

Leitura e escritas digitais são extremamente simples, pois este é o formato de dados que a *Raspberry Pi* utiliza nativamente, portanto a entrada e saída de dados digitais baseia-se em verificar o estado de uma porta ou escrever um sinal alto ou baixo.

Para utilizar dados analógicos é necessário *hardware* externo e desenvolver soluções de *software* mais complexas, apresentadas na figura 6. O conversor Analógico-Digital MCP3008 devolve um valor percentual baseado na diferença potencial nele aplicada, e portanto não converte o valor diretamente, e esta saída possui problemas de precisão, e sofre constantemente de flutuações pois o chip MCP3008 retorna 10 *bits* mais significativos. Já a escrita de dados analógicos, para não ser necessário um chip conversor, foi baseada em Modulação de Largura de Pulso (*PWM*), que têm efeito parecido com saídas analógicas ao simular dados analógicos por meio de entradas altas e baixas intermitentes em uma frequência estabelecida.

O *Raspberry Pi* 3 possui três portas preparadas para *PWM*. Além da porta moduladora de largura de pulso é necessária uma comunicação síncrona serial *full-duplex* utilizando arquitetura mestre-escravo, a qual exige outras três portas (MISO, MOSI, SCLK), e o *Raspberry Pi* possui uma de cada, limitando a saída analógica por PWM a um único periférico.

**4. Conclusão e Discussão**

Figura 6- Funções de escrita e leitura digitais. (Desenvolvida pelo Autor)

Com a aplicação desenvolvida foi possível utilizar um servidor *Node.js* para controlar periféricos de *Raspberry Pi* remotamente. Por conta do *hardware,* a utilização de dados analógicos pelo *Raspberry Pi* é limitada. A entrada e saída de dados analógicos exige *hardware* externo. Por não possuir entradas ou saídas de dados analógicos, foi necessário pesquisar outros meios de se ligar sensores e controladores analógicos ao Raspberry Pi. A solução sugerida por Reck & Sreenivas (2016) é utilizar um conversor analógico digital MCP3002 ou MCP3008. Devido a indisponibilidade do MCP3002 nas lojas de eletrônica locais, foi escolhido o MCP3008.

Utilizar *PWM* não é o modo mais adequado de se enviar informações a um sensor analógico, pois este é suficiente apenas quando a informação não necessita ser quantificada, servindo somente para controlar intensidade de saídas como o *piezo buzzer.* A solução mais adequada para saída de dados analógicos é a utilização de um Circuito Integrado Conversor Digital-Analógico. Devido a não disponibilidade deste circuito integrado em lojas de eletrônica locais, não é possível garantir o funcionamento de saídas analógicas precisas utilizando o *Raspberry Pi*. Para isso, pretende-se realizar pesquisas para que seja verificado o funcionamento do conversor digital-analógico junto ao *Raspberry Pi.*

Por não possuir conexões analógicas, o uso do *Raspberry Pi* para monitoramento de dados analógicos talvez não seja a melhor escolha quando não há conversores digitais-analógicos e analógicos-digitais disponíveis, ou quando a alta precisão dos dados utilizados seja um importante. Ainda assim, devido à grande variedade de extensões, sua alta performance e ser de fácil programação, *Node.js* é um ótimo servidor *web* para o controle remoto de periféricos conectados ao *Raspberry Pi.*

Agradecimentos

Ao Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso – Campus Cuiabá “Octayde Jorge da Silva” pela oportunidade de ingresso neste projeto de pesquisa e extensão, e pelo espaço cedido para a pesquisa. À Fundação de Amparo à Pesquisa do Estado de Mato Grosso pelo apoio financeiro dado, o qual permitiu adquirir todos os materiais utilizados neste projeto.

Referências

CROTTI *et al.* (2013). “Raspberry Pi e Experimentação Remota”, In: International Conference on Interactive Computer aided Blended Learning, 2013 P. 143-145.

MAKSIMOVIĆ *et al.* (2014). “*Raspberry Pi as Internet of Things hardware: Performances and Constraints*”, In: *International Conference on Electrical, Electronic and Computing Engineering*, 1, 2014.

MCCUNE, Robert Ryan (2009). “*Node.js Paradigms and Benchmarks*”. *University of Notre Dame*.

NETCRAFT. “*January 2017 Web Server Survey*”. Disponível em: <https://news.netcraft.com/archives/2017/01/12/january-2017-web-server-survey.html>. Acesso em: 08 mar. 2017.

RIBEIRO JR., Francisco (2012). “Programação Orientada a Eventos no lado do servidor utilizando Node.js”. IFactory Solutions.

RECK, Rebecca M. & SREENIVAS, R. S (2016). “*Developing an Affordable and Portable Control Systems Laboratory Kit with a Raspberry Pi*”. In: *Electronics* 2016, *5th Edition*.

VUJOVIĆ, Vladimir & MAKSIMOVIĆ, Mirjana (2014). “*Raspberry Pi as a Wireless Sensor Node: Performances and Constraints*”, In: *International convention on information and communication technology, electronics and microelectronics*, 37, 2014. P. 1247-1252.