Gerenciamento de energia, multitarefa e conectividade

Sistemas embarcados

Prof. Allan Rodrigo Leite

Reduzindo o *clock* do processador

- É possível reduzir o *clock* do processador do Arduino
  - O *clock* original do Arduino UNO e maioria é de 16Mhz
  - Para isto, recomenda-se o uso de bibliotecas como o Prescaler
    - http://playground.arduino.cc/Code/Prescaler

- Além de reduzir o *clock*, a biblioteca fornece funções adicionais
  - Por exemplo, funções equivalentes à `millis()` e `delay()`
  - Estas funções originais sofrem retardos quando o *clock* é alterado

Reduzindo o *clock* do processador

Constante	Frequência equivalente	Corrente aproximada (com LED da placa desligado)
`CLOCK_PRESCALER_1`	16 MHz	7,8 mA
`CLOCK_PRESCALER_2`	8 MHz	5,4 mA
`CLOCK_PRESCALER_4`	4 MHz	4,0 mA
`CLOCK_PRESCALER_8`	2 MHz	3,2 mA
`CLOCK_PRESCALER_16`	1 MHz	2,6 mA
`CLOCK_PRESCALER_32`	500 kHz	2,3 mA
`CLOCK_PRESCALER_64`	250 kHz	2,2 mA
`CLOCK_PRESCALER_128`	125 kHz	2,1 mA
`CLOCK_PRESCALER_256`	62,5 kHz	2,1 mA

Reduzindo o *clock* do processador

- Exemplo reduzindo o *clock* do processador

```
#include <Prescaler.h>

void setup() {
  pinMode(13, OUTPUT);
  setClockPrescaler(CLOCK_PRESCALER_256);
}

void loop() {
  digitalWrite(13, HIGH);
  trueDelay(1000);

  digitalWrite(13, LOW);
  trueDelay(5000);
}
```

Desabilitando recursos

- É comum desabilitar recursos não utilizados para poupar energia
  - Sobretudo em sistemas embarcados alimentados por baterias
  - Exemplos de recursos
    - Portas SPI
    - Portas I2C
    - Temporizadores
    - Porta serial
    - Entradas analógicas

- Com este gerenciamento é possível poupar energia ao:
  - Ativar um recurso antes do seu uso eventual
  - Utilizar o recurso
  - Depois desativá-lo novamente

Desabilitando recursos

- Para desabilitar recursos é utilizada a biblioteca `Power`
  - Dispõe de um conjunto de funções para desativar os principais recursos

Função	**Descrição**
`power_adc_disable()`	Desabilita as entradas analógicas
`power_spi_disable()`	Desabilita a interface SPI
`power_twi_disable()`	Desabilita TWI (I2C)
`power_usart0_disable()`	Desabilita a UART serial (comunicação serial USB)
`power_timer0_disable()`	Desabilita o temporizador 0 (usado pelas funções millis e delay)
`power_timer1_disable()`	Desabilita o temporizador 1
`power_timer_2_disable()`	Desabilita o temporizador 2
`power_all_disable()`	Desabilita todos os módulos acima.

Desabilitando recursos

- Exemplo de uso da biblioteca Power

```
#include <avr/power.h>

void setup() {
  power_adc_disable();
  power_spi_disable();
  power_twi_disable();
  power_usart0_disable();
  power_timer0_disable();
  power_timer1_disable();
  power_timer2_disable();
  //power_all_disable();
}

void loop() {
}
```

Modo de suspensão

- O Arduino também possui um modo de espera ou suspensão
  - Assim como em computadores

- Para usar esta função recomenda-se a biblioteca `Narcoleptic`
  - https://code.google.com/p/narcoleptic

  - A biblioteca fornece uma função alternativa delay
    - Esta função coloca o Arduino em modo adormecido por um período de tempo
    - É possível adormecer o microcontrolador por um tempo entre as verificações

Modo de suspensão

- Exemplo de uso do modo de suspensão

```
#include <Narcoleptic.h>

void setup() {
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);
    Narcoleptic.delay(1000);

    digitalWrite(13, LOW);
    Narcoleptic.delay(10000);
}
```

Despertando com interrupções externas

- Também é possível manter o microcontrolador adormecido e acordá-lo apenas quando ocorrer alguma interrupção externa
  - Isto permite economizar energia e ativar alguma função apenas quando um sensor detectar alguma alteração

- Esta função é realizada pela biblioteca `Sleep`

Despertando com interrupções externas

- Exemplo despertando com interrupções externas

```
#include <avr/sleep.h>

int pinoLed = 13;
int pinoBotao = 2;

volatile boolean ativo;

void setup() {
    pinMode(pinoLed, OUTPUT);
    pinMode(pinoBotao, INPUT_PULLUP);

    adormecer();
}
```

Despertando com interrupções externas

- Exemplo despertando com interrupções externas (cont.)

```
void loop() {
    if (ativo) {
        ativaLed();
        ativo = false;
        adormecer();
    }
}

void despertar() {
    ativo = true;
}
```

Despertando com interrupções externas

- Exemplo despertando com interrupções externas (cont.)

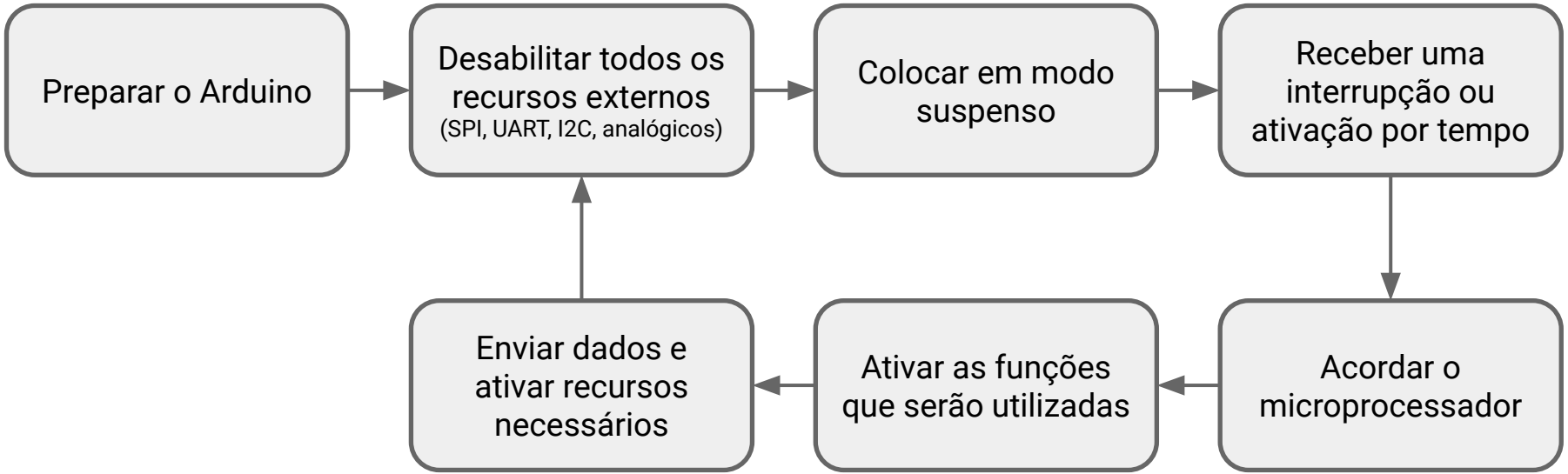
```
void adormecer() {
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_enable();
    attachInterrupt(0, despertar, LOW); //pino D2

    //entra no modo de espera até ser interrompido (LOW)
    sleep_mode();

    //continuará a partir daqui depois de executar a ISR
    sleep_disable();
    detachInterrupt(0);
}
```

Despertando com interrupções externas

- Exemplo despertando com interrupções externas (cont.)

```
void ativaLed() {
    for (int i = 0; i < 20; i++) {
        digitalWrite(pinoLed, HIGH);
        delay(200);

        digitalWrite(pinoLed, LOW);
        delay(200);
    }
}
```

Fluxo para gerenciamento de energia



Exercícios

1. Efetue medições do consumo de energia do Arduino nas condições abaixo. O Arduino deve ser alimentado com a fonte externa na porta VIN (com 7V) e GND. A medição deve ser realizada a partir da ligação de um multímetro em série com o Arduino no modo de medição de corrente (ampere).
    - Normal com loop vazio;
    - Frequência de 1Mhz (`CLOCK_PRESCALER_16`);
    - Menor frequência possível (`CLOCK_PRESCALER_256`);
    - Todos os componentes de hardware desligados; e
    - Adormecido.

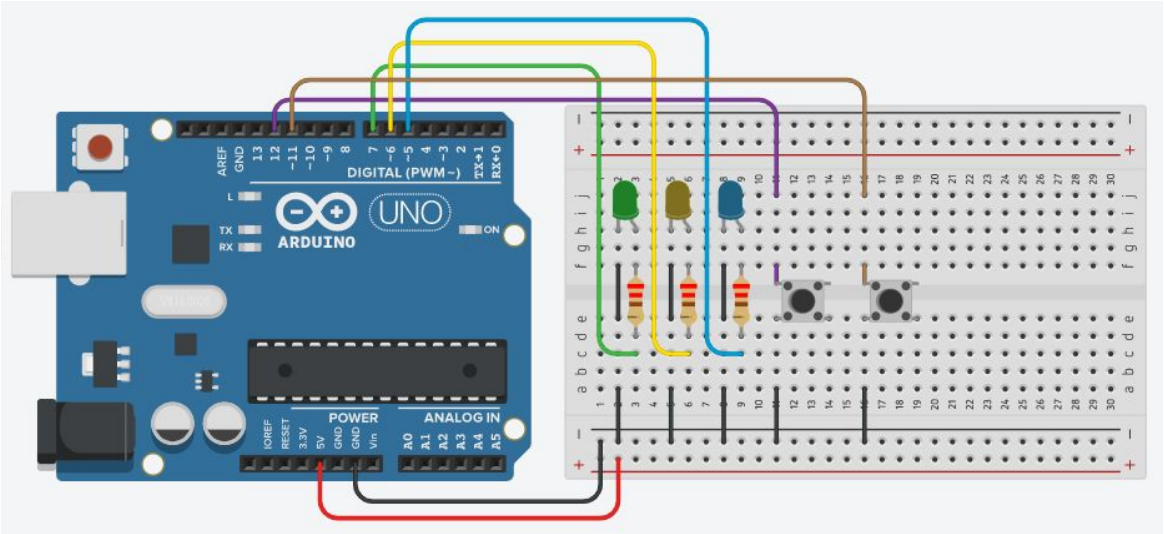
Multitarefas no Arduino

- O `delay()` é uma função blocante
  - Isto significa que o Arduino ficará travado durante a execução da função
  - Por isto, recomenda-se evitar o uso desta função na função `loop()`
  - A exceção do uso do `delay()` é na função `setup()`
    - Geralmente o `delay()` é usado para configuração de alguma biblioteca

- Maneiras para evitar o uso do `delay()`
  - A interrupção é uma das formas de evitar o uso desta função
  - Outra forma é utilizar a função `millis()`

Multitarefas no Arduino

- Exemplo de multitarefas
  - Crie um sketch no Arduino para controlar três LEDs
    - O primeiro deve piscar a cada segundo
    - Os demais LEDs devem ser acionados por *push buttons*



Multitarefas no Arduino

- Exemplo de multitarefas (cont.)

```
int pinoBotaoAmarelo = 12;
int pinoBotaoAzul = 11;

int pinoLedVerde = 7;
int pinoLedAmarelo = 6;
int pinoLedAzul = 5;

void setup() {
    pinMode(pinoLedVerde, OUTPUT);
    pinMode(pinoLedAmarelo, OUTPUT);
    pinMode(pinoLedAzul, OUTPUT);

    pinMode(pinoBotaoAmarelo, INPUT_PULLUP);
    pinMode(pinoBotaoAzul, INPUT_PULLUP);
}
```

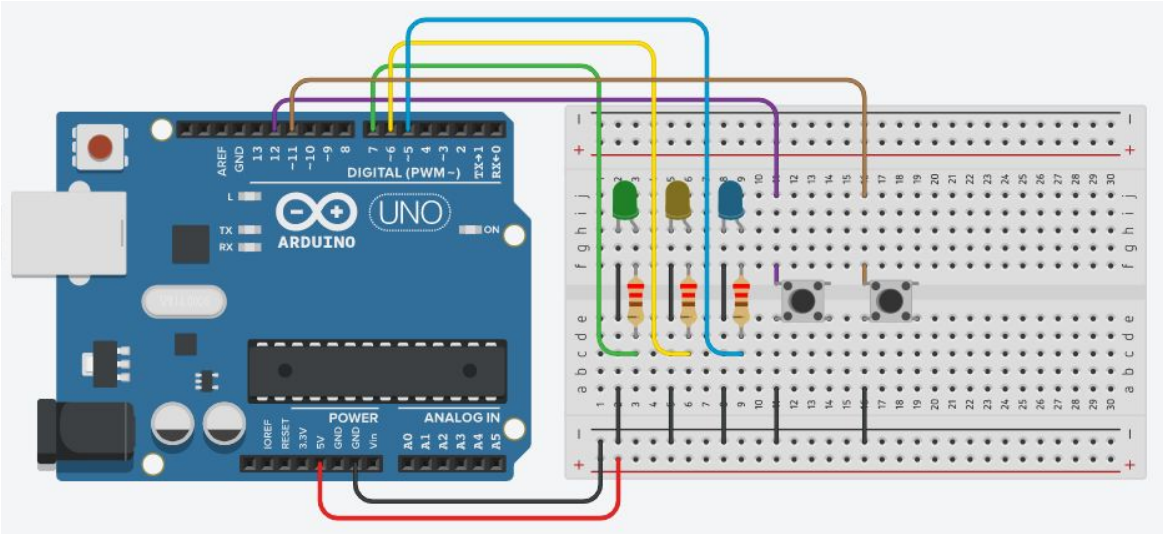
Multitarefas no Arduino

- Exemplo de multitarefas (cont.)

```
void loop() {
    digitalWrite(pinoLedVerde, HIGH);
    delay(1000);

    digitalWrite(pinoLedVerde, LOW);
    delay(1000);

    int amarelo = digitalRead(pinoBotaoAmarelo);
    digitalWrite(pinoLedAmarelo, amarelo == LOW ? HIGH : LOW);

    int azul = digitalRead(pinoBotaoAzul);
    digitalWrite(pinoLedAzul, azul == LOW ? HIGH : LOW);
}
```

Multitarefas no Arduino

- Exemplo de multitarefas utilizando a função `millis()`
  - Crie um sketch no Arduino para controlar três LEDs
    - O primeiro deve piscar a cada segundo
    - Os demais LEDs devem ser acionados por *push buttons*



Multitarefas no Arduino

- Exemplo de multitarefas utilizando a função `millis()` (cont.)

```
int pinoBotaoAmarelo = 12;
int pinoBotaoAzul = 11;

int pinoLedVerde = 7;
int pinoLedAmarelo = 6;
int pinoLedAzul = 5;

void setup() {
    pinMode(pinoLedVerde, OUTPUT);
    pinMode(pinoLedAmarelo, OUTPUT);
    pinMode(pinoLedAzul, OUTPUT);

    pinMode(pinoBotaoAmarelo, INPUT_PULLUP);
    pinMode(pinoBotaoAzul, INPUT_PULLUP);
}
```

Multitarefas no Arduino

- Exemplo de multitarefas (cont.)

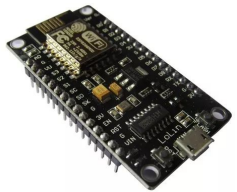
```
void loop() {
    static unsigned long tempo;

    digitalWrite(pinoLedVerde, millis() - tempo < 1000 ? HIGH : LOW);

    if (millis() - tempo >= 2000)
        tempo = millis();

    int amarelo = digitalRead(pinoBotaoAmarelo);
    digitalWrite(pinoLedAmarelo, amarelo == LOW ? HIGH : LOW);

    int azul = digitalRead(pinoBotaoAzul);
    digitalWrite(pinoLedAzul, azul == LOW ? HIGH : LOW);
}
```

Exercícios

2. Crie um sketch no Arduino para controlar cinco LEDs. Os três primeiros deve acender a cada um, dois e três segundos, respectivamente. Contudo, os três LEDs devem ser apagados simultaneamente. Os demais LEDs devem ser acionados imediatamente por *push buttons*. Por fim, deve existir um *push button* adicional para iniciar a sequência de acionamento dos três primeiros LEDs a qualquer momento.

Conectividade e WiFi

- O ESP8266 é um microcontrolador com um módulo WiFi integrado
  - O módulo, também chamado de NodeMCU suporta o protocolo TCP/IP
  - Permite que o microcontrolador acesse redes wifi para
    - Prover aplicações baseadas no protocolo TCP/IP
    - Consumir recursos de uma rede wifi
  - Para projetos Arduino é necessário o uso de *shield* WiFi



Módulo ESP8266

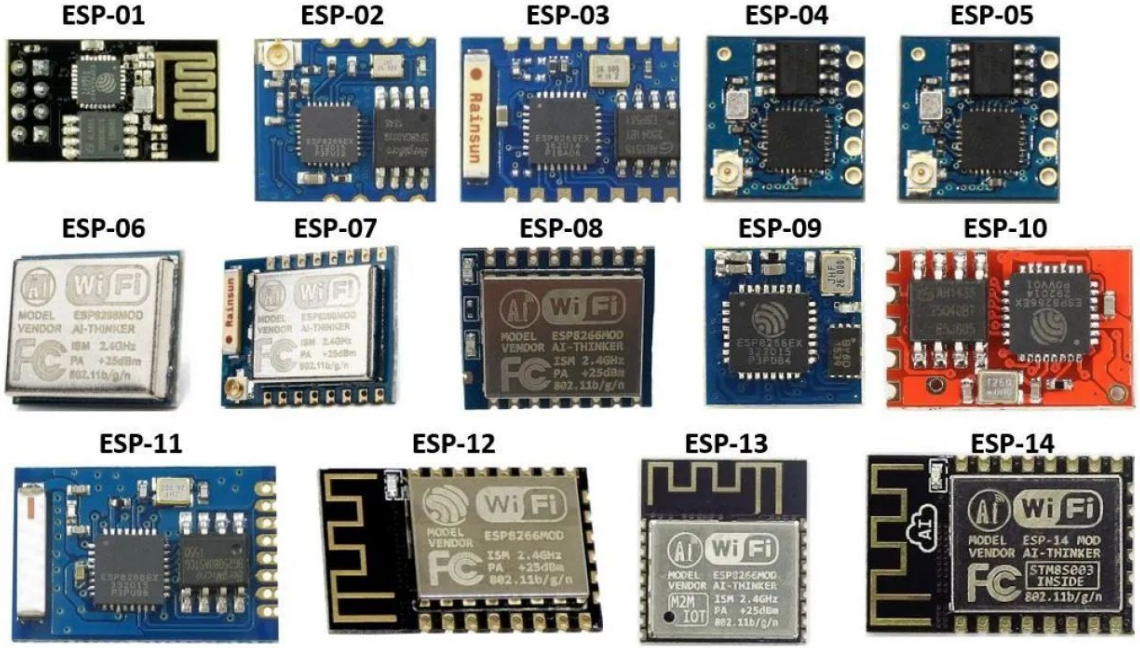
Arduino + *shield* wifi

Conectividade e WiFi

- Características do ESP8266
  - Microcontrolador com wifi embutido padrão IEEE 802.11 B/G/N
  - Alcance aproximado de 90 metros
  - Tensão de operação de 3,3 V (DC)
  - CPU com 80 Mhz e pode operar com até 160 Mhz
  - Arquitetura RISC de 32 bits
  - 32 KB de RAM para instruções e 96 KB de RAM para dados
  - 64 KB de ROM para *boot*
  - Modos de operação
    - Cliente
    - *Access point*
    - Cliente + *access point*

Conectividade e WiFi

- Tipos de ESP8266



Conectividade e WiFi

- Usos do ESP8266
  - ESP8266 como ponte serial WiFi para um Arduino
    - Usa as portas RX/TX para operar como placa de comunicação
  - ESP8266 *standalone*
    - O ESP8266 é um microcontrolador, portanto, a aplicação pode executar nele

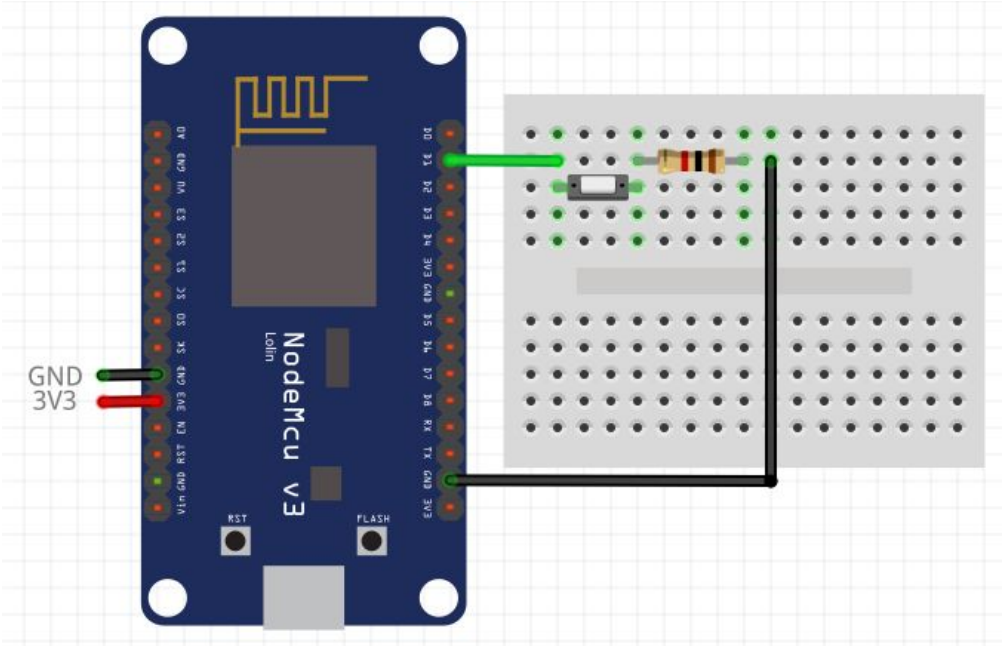
- Linguagens de programação suportadas
  - Luascript (disponível por padrão no firmware do NodeMCU)
  - C/C++
    - Permite reusar bibliotecas do Arduino na maioria dos casos
    - Pode ser utilizado a mesma IDE do Arduino com *plugin* para NodeMCU
  - MicroPython
  - Javascript

Configurando o ESP8266 na IDE do Arduino

- Para configurar a biblioteca do ESP8266 é necessário
  - Preferências > URLS adicionais para gerenciadores de placas
    - http://arduino.esp8266.com/stable/package_esp8266com_index.json
  - Ferramentas > Placa > Gerenciador de placas
    - Informe ESP8266 e instale o pacote “ESP8266 by ESP8266 Community”
  - Ferramentas > Placa
    - Selecione o ESP8266 Module

Primeira aplicação em ESP8266

- Acionar o LED *built-in* a partir de um *push button*



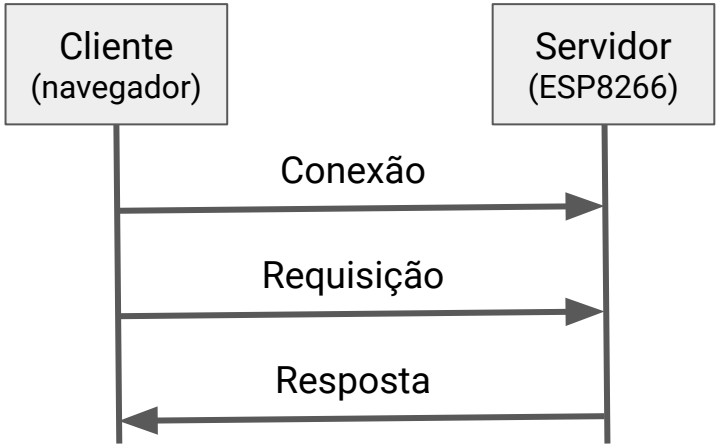
Primeira aplicação em ESP8266

- Acionar o LED *bult-in* a partir de um *push button* (cont.)

```
int pinoBotao = D1;

void setup() {
    pinMode(pinoBotao, INPUT);

    //LED_BUILTIN é uma constante para o LED embarcado do ESP8266
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_BUILTIN, digitalRead(pinoBotao) == HIGH ? LOW : HIGH;
}
```

Conectividade com um ESP8266

- É possível prover um HTTP server com biblioteca `ESP8266WiFi.h`
  - Esta biblioteca possui funções para
    - Configuração do SSID (identificador da rede) e senha (senha da rede)
    - Criação do HTTP server em uma porta predeterminada
    - Identificação de novas requisições HTTP (cliente)
    - Recebimento e envio dos dados (requisição e resposta)



Conectividade com um ESP8266

- Principais funções da biblioteca `ESP8266WiFi.h`
  - `WiFi.begin(<ssid>, <senha>)`
    - Conecta em uma rede WiFi usando o identificador e senha informados
  - `WiFi.localIP()`
    - Retorna o IP local (cliente)
  - `WiFi.status()`
    - Retorna a situação da tentativa de conexão na rede WiFi configurada
    - `WL_CONNECTED`: conexão realizada com sucesso
    - `WL_CONNECT_FAILED`: falha na conexão
    - `WL_NO_SSID_AVAIL`: SSID não está disponível
    - `WL_CONNECTION_LOST`: conexão perdida
    - `WL_DISCONNECTED`: desconexão da rede

Conectividade com um ESP8266

- Principais funções da biblioteca `ESP8266WiFi.h` (cont.)
  - `WiFiServer server(<porta>)`
    - Cria uma instância de um HTTP server na porta definida
  - `WiFiServer.begin()`
    - Inicia o HTTP server
  - `WiFiServer.available()`
    - Verifica se algum cliente se conectou ao server
    - A função retornará um `WiFiClient` (representação do cliente)

Conectividade com um ESP8266

- Principais funções da biblioteca `ESP8266WiFi.h` (cont.)
  - `WiFiClient.available()`
    - Verifica o cliente está conectado
  - `WifiClient.flush()`
    - Descarta todos os dados não lidos que foram enviados do cliente
  - `WifiClient.println(<texto>)`
    - Envia dados ao cliente

Conectividade com um ESP8266

- Exemplo de conexão em rede WiFi e HTTP server

```
#include <ESP8266WiFi.h>

const char* ssid = "nome da rede WiFi";
const char* senha = "senha da rede WiFi";


WiFiServer server(80); //cria uma instância de servidor na porta 80
```

Conectividade com um ESP8266

- Exemplo de conexão em rede WiFi e HTTP server (cont.)

```
void setup() {
    Serial.begin(9600);
    Serial.print("Conectando a ");
    Serial.println(ssid);

    WiFi.begin(ssid, senha); //tenta conectar na rede WiFi informada

    while (WiFi.status() != WL_CONNECTED) delay(500); //aguarda até conectar

    Serial.println("Conectado!");
    server.begin(); //inicia o HTTP server na porta especificada

    Serial.print("Servidor iniciado no endereço http://");
    Serial.println(WiFi.localIP());
}
```

Conectividade com um ESP8266

- Exemplo de conexão em rede WiFi e HTTP server (cont.)

```
void loop() {
    WiFiClient client = server.available(); //verifica cliente conectado
    if (!client) return; //se não houver, encerra a execução do loop

    Serial.println("Novo cliente conectado!");
    while (!client.available()) delay(10); //aguarda até estar disponível

    String request = client.readStringUntil('\r'); //lê a primeira linha
    Serial.println(request);

    client.flush(); //descarta os demais bytes não lidos
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println("<!DOCTYPE HTML><html><h1>Olá mundo!</h1></html>");
}
```

Conectividade com um ESP8266

- Exemplo de conexão em rede WiFi e HTTP server (cont.)

Conectividade com um ESP8266

- Consumo de serviços com a biblioteca `ESP8266HTTPClient.h`
  - Esta biblioteca permite realizar requisições HTTP a partir do ESP8266
  - Neste caso, o ESP8266 será o cliente da requisição
  - Já o serviço a ser consumido deve estar acessível a partir da rede WiFi

- Esta biblioteca fornece uma abstração de requisição HTTP
  - Esta abstração é representada pela classe `HTTPClient`

Conectividade com um ESP8266

- Principais métodos da classe `HTTPClient`
  - `HTTPClient.begin(<URL>)`
    - Especifica o endereço (URL) da requisição a ser realizada
  - `HTTPClient.GET()`
    - Realiza a requisição HTTP utilizando o verbo GET
    - O retorno deste método é um HTTP status code
    - `200`: OK
    - `201`: criado
    - `202`: aceito
    - `400`: requisição inválida
    - `401`: não autorizado
    - `500`: erro interno

Conectividade com um ESP8266

- Principais métodos da classe `HTTPClient`
  - `HTTPClient.POST(<payload>)`
    - Realiza a requisição HTTP utilizando o verbo POST
    - Possibilita informar dados (*payload*) na requisição
  - `HTTPClient.addHeader(<chave>, <valor>)`
    - Permite configurar parâmetros no cabeçalho da requisição HTTP
    - `HTTPClient.addHeader("Content-Type","text/plain")`
  - `HTTPClient.getString()`
    - Retorna os dados (*payload*) da requisição, ou seja, a resposta da requisição
  - `HTTPClient.end()`
    - Finaliza a conexão HTTP estabelecida

Conectividade com um ESP8266

- Exemplo de requisição HTTP

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClientSecure.h>

const char* ssid = "nome da rede WiFi";
const char* senha = "senha da rede WiFi";
```

Conectividade com um ESP8266

- Exemplo de requisição HTTP (cont.)

```
void setup() {
    Serial.begin(9600);
    Serial.print("Conectando a ");
    Serial.println(ssid);

    WiFi.begin(ssid, senha); //tenta conectar na rede WiFi informada

    while (WiFi.status() != WL_CONNECTED) { //aguarda até conectar
        delay(500);
    }

    Serial.println("Conectado!");
}
```

Conectividade com um ESP8266

- Exemplo de requisição HTTP (cont.)

```
void loop() {
    HTTPClient http;
    http.begin("https://reqres.in/api/users/2"); //informa a URL

    int codigo = http.GET(); //Realiza a requisição HTTP

    if (codigo == 200) { //resposta OK
        Serial.println("Resposta da requisição");
        Serial.println(http.getString());
    }

    http.end(); //encerra a conexão
    delay(5000);
}
```

Exercícios

3. Crie um sketch no ESP8266 para realizar a leitura em um sensor de temperatura e umidade. Os dados da leitura devem ser disponibilizados em uma página HTML. Para isto, o ESP8266 deve utilizar uma rede WiFi já existente para conexão, bem como disponibilizar um HTTP server para ser realizada a visualização dos dados da leitura.

4. Crie um sketch no ESP8266 para consumir os dados de um serviço web e disponibilizá-los em uma página HTML. Para tal, o ESP8266 deve utilizar uma rede WiFi já existente para conexão, consumir um serviço web disponível e disponibilizar um HTTP server para ser realizada a visualização dos dados retornados do serviço web.
    - Utilize o endereço https://reqres.in/api/users/2 para os testes.

5. Refatore o código do exercício 4 para utilizar a biblioteca `ArduinoJson.h` para facilitar a manipulação dos dados no formato JSON.

6. Refatore o código do exercício 3 para gravar as leituras do sensor de temperatura e umidade em um serviço web. O serviço web deve fornecer um *end-point* para receber os dados e armazená-los em um banco de dados, bem como o horário da leitura.

Exercícios

7. Crie um sketch no ESP8266 para realizar criar detectar antenas WiFi próximas. O ESP8266 deve atuar como um sniffer, a fim de detectar os dispositivos WiFi próximos e exibir na saída serial os respectivos endereços MAC dos dispositivos, bem como a intensidade do sinal. Para isto, utilize o modo promíscuo a partir das funções abaixo disponíveis na biblioteca `ESP8266WiFi.h`.
   - `wifi_set_opmode(STATION_MODE);`
   - `wifi_set_channel(<canal WiFi>);`
   - `wifi_promiscuous_enable(WIFI_P_DISABLE | WIFI_P_ENABLE);`
   - `wifi_set_promiscuous_rx_cb(<callback>);`

8. Crie um sketch no ESP8266 para prover um *access point* configurável para gerenciar conexões WiFi. O ESP8266 deve iniciar um portal de configuração WiFi e salvar os dados em memória. Em seguida, o ESP8266 deve conectar-se em modo estação na rede WiFi selecionada a partir das configurações realizadas. Para tal, utilize a classe `WiFiManager` disponível na biblioteca `WiFi.h`.

Gerenciamento de energia, multitarefa e conectividade

Sistemas embarcados

Prof. Allan Rodrigo Leite