nodeMCU - comunicação

troca de mensagens

- send (destino, msg)

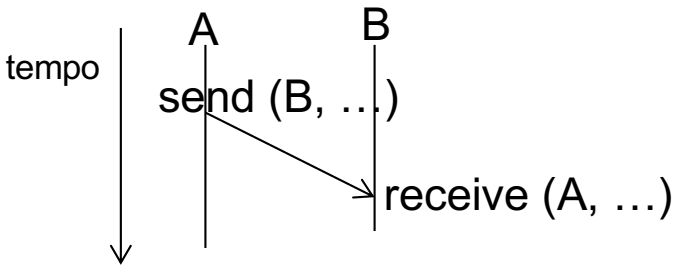
- receive (origem, mensagem)

  - questões

    - semântica de operações
    - especificação de origem e destino
    - formato de mensagem

envio síncrono e assíncrono

- envio assíncrono: execução procede imediatamente



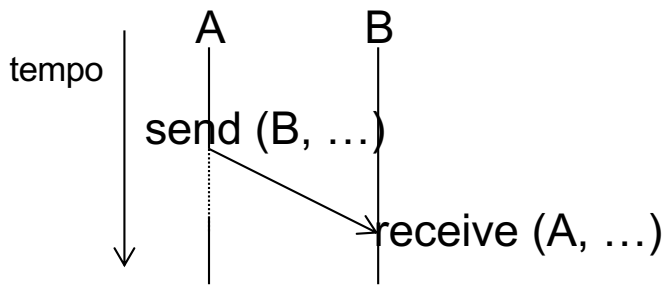
- bufferização
- concorrência
- determinismo

alternativas ao bloqueio

- orientação a eventos - recebimento implícito
  - é o que usaremos no nodemcu
- outras alternativas: futuros

envio síncrono e assíncrono

- envio síncrono: execução só procede quando destinatário recebe msg



- bufferização
- concorrência
- determinismo

especificação de origem e destino

- como identificar parceiro de comunicação
  - identificadores específicos
  - serviços de nomes
  - caixas de correio
  - o exemplo < IP, porta >

formato de mensagens

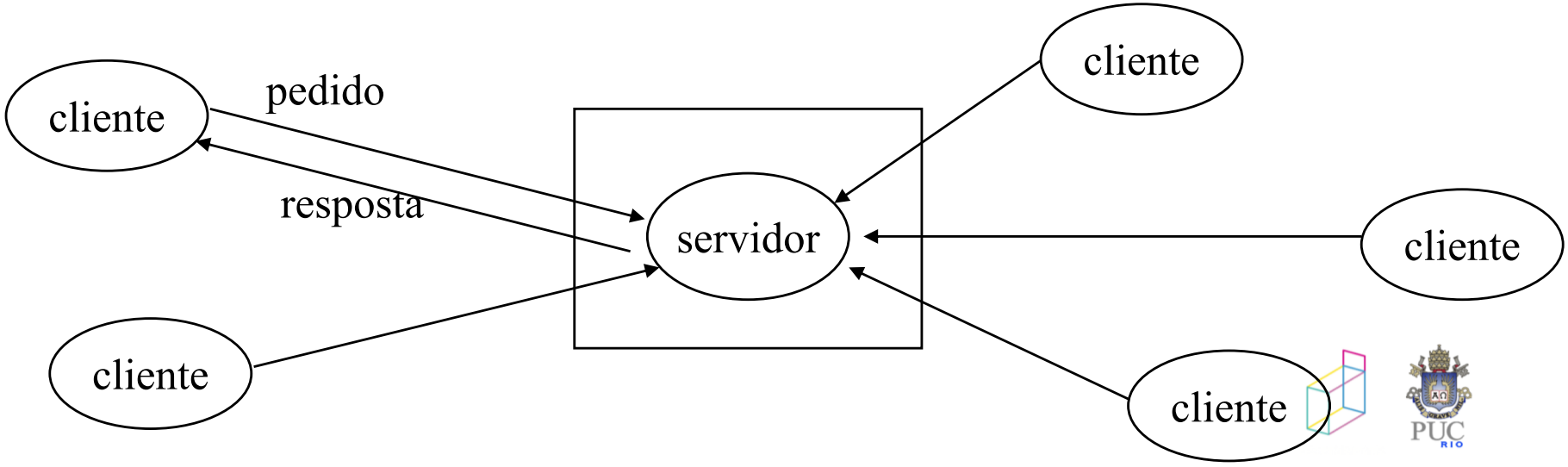
- parte do *protocolo* de comunicação
- protocolos binários e textuais
  - conversão entre representações

arquiteturas de sistemas

- (paradigmas, padrões, ...)
- necessidade de organizar a comunicação para entender o programa distribuídos
- arquiteturas:
  - centralizadas: cliente-servidor
  - decentralizadas: p2p, filtros, pub/sub...

Cliente-Servidor

- Modelo mais usado para aplicações distribuídas não paralelas;
- Um processo *servidor* está sempre a espera de comunicação;
- O processo *cliente* tem a iniciativa de começar a comunicação quando deseja algum serviço.



comunicação no nodemcu

- módulo wifi
- comunicação básica (sockets) e bibs + alto nível
  - http (cliente-servidor)
  - mqtt (pub/sub)

conexão wifi

```
-- Conexao: criando rede de wifi nova
wifi.setmode(wifi.SOFTAP)
wifi.ap.config({ssid="nodemcu",pwd="11112222"})
wifi.ap.setip({ip="192.168.0.20",netmask="255.255.255.0",
                gateway="192.168.0.20"})
print(wifi.ap.getip())
```

```
-- Conexão: conectando-se a uma rede existente
wificonf = {
  -- verificar ssid e senha
  ssid = "reativos",
  pwd = "reativos",
  got_ip_cb = function (con)
                print ("meu IP: ", con.IP)
              end,
  save = false}

wifi.setmode(wifi.STATION)
wifi.sta.config(wificonf)
```


api nodeMCU

- API toda baseada em callbacks

- callbacks registradas através de fçs específicas

```
srv=net.createServer(net.TCP) – bloqueante
if srv then
  srv:listen(80,
    function(conn)
    -- fç chamada qdo cliente conectar
       conn:on("receive",
         function(client,request)
         -- fç chamada qdo receber algo na conexão
           print("RECEIVE!")
         end)
       end)
```

HTTP

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
…
```

```
GET /path/script.cgi?field1=value1&field2=value2 HTTP/1.0
```

- protocolo sem conexão e sem estado

nodeMCU

- podemos usar fçs de manipulação de strings de Lua para "parsear" os pedidos http:

```
srv=net.createServer(net.TCP)

if srv then
  srv:listen(80,function(conn)
        conn:on("receive",
          function(client,request)
            local _, _, method, path, vars =
string.find(request,
    "([A-Z]+) ([^?]+)%?([^ ]+) HTTP");
```

GET /path/script.cgi?field1=value1&field2=value2 HTTP/1.0

cliente-servidor http

```
srv:listen(80,
  function(conn)
    conn:on("receive",
      function(client,request)
          local _, _, method, path, vars =
            string.find(request, …)
          -- executa requisição
          buf = [[
            <h1><u>PUC Rio</u></h1>
            <h2><i>ESP8266 Web Server</i></h2>";
              … restante da página html
          ]]
          sck:send(buf, function()
            print("respondeu") sck:close()
          end)
      end)
    end)
```

http://www.inf.puc-rio.br/~noemi/sr-19/code/nodemcu/servidorpisca.lua

filas de mensagens e pub/sub

- servidor de filas (broker)
  - filtragem de mensagens
  - segurança
  - disponibilidade
  - persistência

protocolo mqtt

- protocolo binário "leve"
- identificadores de filas são strings
- mensagens são strings (podem conter qq coisa)
- implementação para nodeMCU

protocolo mqtt no nodeMCU

```
m = mqtt.Client("clientid”, 120)
-- conecta com servidor mqtt na máquina
'ipbroker' e porta 1883:
m:connect(ipbroker, 1883, 0,
  -- callback em caso de sucesso
  function(client) print("connected") end,
  -- callback em caso de falha
  function(client, reason)
    print("failed reason: "..reason)
  end)
```

protocolo mqtt no nodeMCU (2)

```
m = mqtt.Client("clientid", 120)

-- conecta com servidor mqtt na máquina
'ipbroker' e porta 1883:
m:connect("test.mosquitto.org", 1883, 0,
  -- callback em caso de sucesso
  function(client) print("connected") end,
  -- callback em caso de falha
  function(client, reason)
    print("failed reason: "..reason)
  end)
```

protocolo mqtt no nodeMCU (3)

```
m:subscribe("alos",0,
        -- fç chamada qdo inscrição ok:
        function (client)
          print("subscribe success")
        end
)

m:on("message",
     function(client, topic, data)
       print(topic .. ":" )
       if data ~= nil then print(data) end
     end
)
```

- tipicamente chamadas dentro de callbacks!

mqtt

- vários servidores disponíveis: mqtt.org
- servidor mosquitto (test.mosquitto.org)
  - servidor
  - programas de linha de comando para consumo e produção de mensagens
- exemplos anteriores em http://www.inf.puc-rio.br/~noemi/sr-19/code/nodemcu/comunicacao/

comunicação com o löve

lua_mqtt

```
mqtt = require("mqtt_library")

mqtt_client = mqtt.client.create(hostname, port, callback)

 function callback(topic, payload)
    -- application specific code
 end

mqtt_client:connect(identifier, ...)

mqtt_client:publish(topic, payload)

mqtt_client:subscribe(topics)

mqtt_client:handler()
```

- exemplo em http://www.inf.puc-rio.br/~noemi/sr-19/code/lovelua/mqtt/main.lua

exercício

- gerar alguma comunicação entre programa löve e nodemcu
  - botão pressionado -> manda msg para löve
  - msg do löve faz acender led

atenção!!!!
1)criar canal usando seu número de matrícula
2)cada programa cliente tem que se conectar com identificador distinto!