# CAMADA FÍSICA DA COMPUTAÇÃO

ENGENHARIA DA COMPUTAÇÃO - Rodrigo Carareto – 0x07E4/02

## **AVALIAÇÃO 1 –** 0x07E4-02

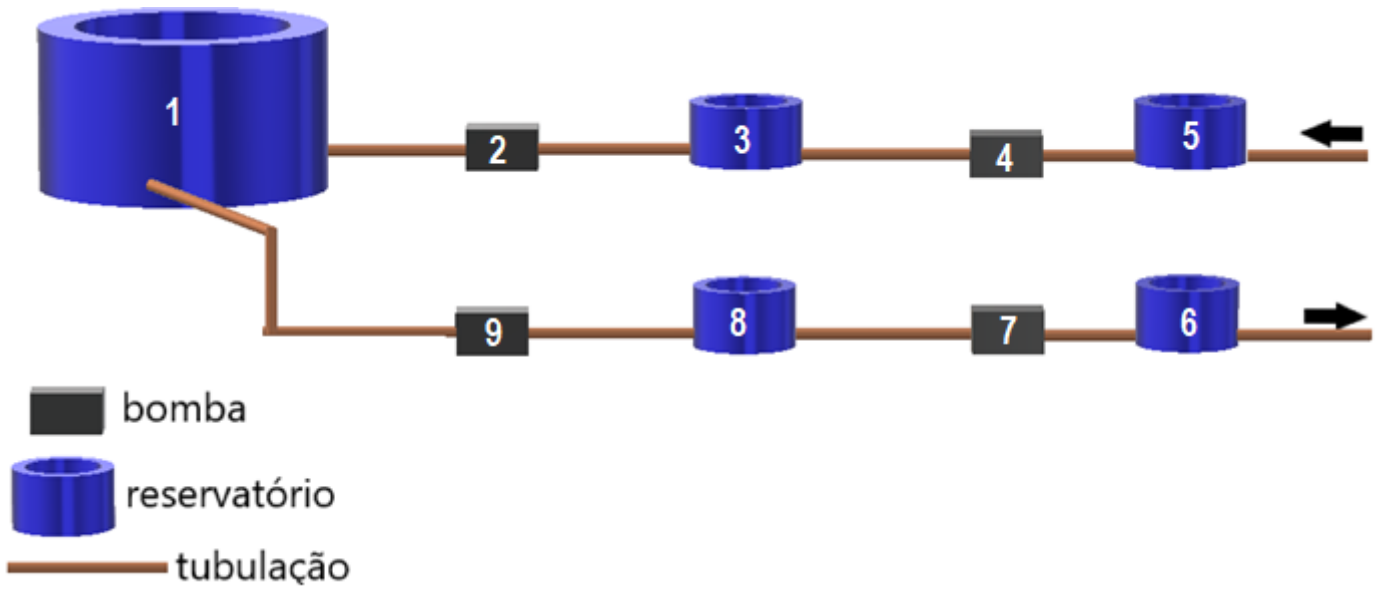
**NOME: Giovanni Santos**

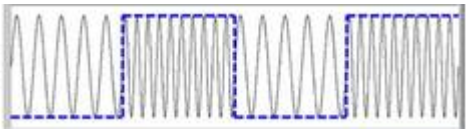
Você pode responder as questões editando esse arquivo e submete-lo (sem a necessidade de redigir e digitalizar o manuscrito).

### Questão 1.

Nos projetos da disciplina utilizamos uma ferramenta de softwares construída em Python para transmissão serial de dados ponto a ponto entre duas aplicações. Para isso, ainda utilizamos Arduinos conectados entre si para termos a comunicação física entre as duas portas seriais do seu computador (alguns alunos emularam as portas através de softwares). Para a possibilidade de envio e recebimentos de "arrays" de bytes, utilizamos ainda funções implementadas em "threads", que atuavam de maneira independente. A aplicação, além de compartilhar uma variável com cada um dos threads, ainda tinha a capacidade de ativar e desativar tais "threads".

Objetivando explicar o funcionamento geral do software para seu colega, um aluno teve a ideia de fazer uma analogia com um sistema hidráulico. Nesse sistema, os reservatórios representavam "buffers", aplicações ou variáveis. As bombas d'água representavam funções. As tubulações, o fluxo de dados entre variáveis. As setas apontam o sentido de chegada e saída da água no sistema, ou seja, o sentido do fluxo de dados.







a) Numere os elementos do software de comunicação serial na lista abaixo de 1 a 9 de acordo com a numeração dos elementos do modelo hidráulico. Associe os elementos do software aos elementos do sistema hidráulico de acordo com as funções de cada elemento de modo a tornar a analogia coerente.

*Aplicação ( 1 )*

*Buffer do chip UART para recebimento de dados ( 5 )*

*Buffer do chip UART para envio de dados ( 6 )*

*Variável compartilhada entre aplicação e thread para recebimento de dados ( 3 )*

*Variável compartilhada entre aplicação e thread para envio de dados ( 8 )*

*Thread RX ( 2 )*

*Thread TX ( 9 )*

*Método sendData ( 7 )*

*Método gerData ( 4 )*

b) Durante o recebimento de dados a função executada em thread foi implementada como mostrado abaixo. Repare que tal função só realiza alguma coisa quando a variável "threadMutex" é verdade ficando inativa quando "threadMutex" é falso. Em que circunstâncias essa variável deve ser fixada como verdade e em que circunstancia em falsa? Que tipo de problema você esperaria ter caso esse controle não fosse feito?

```
25
26      def thread(self):
27          while not self.threadStop:
28              if(self.threadMutex == True):
29                  rxTemp, nRx = self.fisica.read(self.READLEN)
30                  if (nRx > 0):
31                      self.buffer += rxTemp
32                  time.sleep(0.01)
33
```

Rb: Essa variável deve ser fixada como verdade caso haja uma conexão entre as portas Tx e Rx e se thredStop for falso, caso contrário ela deverá ser falsa. Um possível problema causado pela falta desse controle seria que a aplicação iria tentar enviar bytes para outra mesmo ela não estando lá.

c) Repare que no caso do thread para envio, o comando do threadMutex também existe, como mostrado abaixo. Nesse caso, em que circunstância a variável é fixada em verdadeiro e em que circunstância é fixada em falsa? Que tipo de erro você esperaria caso o controle não existisse?

```
27
28      def thread(self):
29          while not self.threadStop:
30              if(self.threadMutex):
31                  self.transLen    = self.fisica.write(self.buffer)
32                  self.threadMutex = False
33
```

Rc: Essa variável é fixada em verdadeiro até que o threadStop seja falso, pois enquanto ele for verdadeiro, junto com o threadMutex, a variável threadMutex será alterada para falso. O thread iria entrar num loop infinito, podendo crashar a aplicação.

## Questão 2.

Uma comunicação ponto a ponto UART está funcionando como um "streaming" de dados com a seguinte configuração feita através da classe "física":

```
15 ##################################
16 # Interface com a camada física #
17 ##################################
18 class fisica(object):
19     def __init__(self, name):
20         self.name        = name
21         self.port        = None
22         self.baudrate    = 115200
23         self.bytesize    = serial.EIGHTBITS
24         self.parity      = serial.PARITY_EVEN
25         self.stop        = serial.STOPBITS_ONE
26         self.timeout     = 0.1
27         self.rxRemain    = b""
28
```

a) Com esta configuração, qual o tempo mínimo possível para a transferência de um arquivo de 1k bytes supondo que foi utilizado para a transmissão a fragmentação através do seguinte datagrama:
   - Head – 8 bytes
   - Payload – 64 bytes
   - EOP – 4 bytes

Ra: 1000/64 = 15,625 logo… ((16 payloads * 12 head+eop) + 1000) bytes * 8 bits = 9536 bits

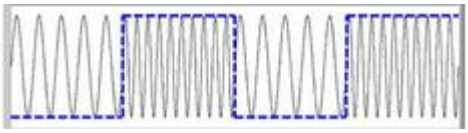
…

x = 9536/115200 = 0,0827 segundos para fazer a transferência de 1k bytes

Levando em conta que cada byte posuíria um bit de paridade e um bit de parade haveriam 2384 bits a mais na conta, fazendo o resultado ser x = 11920/115200 = 0,103 segundos (no caso, estaria excedendo o timeout).

b) Dado que a transmissão de dados utilizando fragmentação em datagramas implica transmissão de um número maior de bytes, em que circunstância a transmissão fragmentada pode ser mais rápida que a não fragmentada, considerando-se que os dados devam ser transmitidos com integridade?

Rb: O único caso que consigo imaginar que a transmissão fragmentada possa ser mais rápida é quando o baudrate dessa transmissão é minimizado e menor que aquela onde a transmissão não é fragmentada, fazendo com que a transmissão seja mais rápida.





## Questão 3)

A função utilizada para em uma camada de enlace para receber dados foi a "getData":

```
41
42      def getData(self, size):
43          data = self.rx.getNData(size)
44          return(data, len(data))
45
```

Esta função utiliza-se da função da camada inferior, "getNdata":

```
70      def getNData(self, size):
71          while(self.getBufferLen() < size):
72              time.sleep(0.05)
73          return(self.getBuffer(size))
74
```

que, por sua vez, utiliza a função "getBuffer":

```
63      def getBuffer(self, nData):
64          self.threadPause()
65          b           = self.buffer[0:nData]
66          self.threadResume()
67          return(b)
68
```

a) A função "getBuffer" mostrada acima, foi ligeiramente modificada. Qual foi a modificação? A função ainda está funcionando corretamente? Justifique.

Ra: Está faltando a seguinte linha:

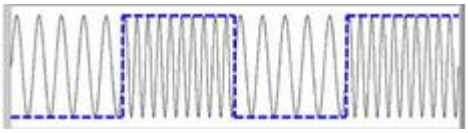
```
self.buffer = self.buffer[nData:]
```

Ela reescreve a variável buffer para o resto além do tamanho pedido pela função. Ela não estará funcionando corretamente pois a variável buffer iria apenas aumentar de acordo com os bits recebidos por ela e toda vez que chamasse essa função ela sempre retornaria os mesmos bits iniciais presentes em buffer.

b) Caso não esteja, que tipo de erro você esperaria ter ao utilizar esta função?

Rb: Assim como dito acima, a variável buffer iria apenas aumentar de tamanho conforme o recebimento de bits e iria retornar sempre os mesmos valores presentes no começo da variável. Ex: se eu chamasse a função getData() com o mesmo tamanho duas vezes, o resultado seria o mesmo.

c) Como corrigir o problema?
Uma maneira para corrigir o problema é adicionando a linha que foi previamente retirada ou até mesmo utilizando a função de clearBuffer() depois de definir a variável b, mas isso geraria outros problemas, como a grande chance de perda de dados.





## Questão 4)

No desenvolvimento de um projeto foi necessária a transmissão de dados entre um sensor de velocidade angular e uma central de processamento e controle.

- O sensor realiza 1k leituras por segundo.
- Cada leitura do sensor é registrada em 16 bits.
- A transmissão deve ser feita serialmente utilizando-se padrão UART em formato streaming, em tempo real, sem a utilização de buffer entre o sensor e a central (toda leitura realizada deve ser instantaneamente transmitida).
- Também não foi utilizado datagrama (pacotes com head e EOP).

Nesse caso, configure sua comunicação UART com o menor baudrate possível para a transmissão descrita.

Lê 16000 bits por segundo

| PARÂMETRO | VALOR ADOTADO | VALORES POSSÍVEIS |
|---|---|---|
| **STOP BITS** | 1 | 1 a 2 bits |
| **PARITY BITS** | 0 | 0 a 1 bit |
| **BAUDRATE** | 18000 | mínimo possível |
| **BYTESIZE** | 8 | 5 a 8 bits |

Como o sensor lê 16000 bits por segundo, quanto maior o bytesize, menor vão ser os bits extras gerados pelos stopbits. No caso, a transmissão teria 16000/8 = 2000 bit a mais nas leituras, logo a taxa de transmissão seria de 16000+2000=18000.