Leitura de porta serial utilizando Python e Raspberry PI

Hoje vou explicar como fazer a leitura de porta serial utilizando Python e Raspberry PI.

Suponhamos que você trabalhe para um cliente que possui uma balança Toledo 820J/XIV, conectada a um display Toledo 8540, ambos fabricados em abril de 2010.

Você quer monitorar, em tempo real, a balança e publicar o valor medido, continuamente, em um webservice. Como fazer isso?

Passo 1 – Obtenha a documentação do fabricante

Sempre que você deseja se comunicar com um produto de terceiros (seja ele um hardware, ou até mesmo um webservice), o primeiro passo é obter a documentação do fabricante, para entender como a comunicação deve ser realizada.

Estudando os manuais dos dois produtos (balança Toledo 820J/XIV e display Toledo 8540), você irá descobrir que o fabricante disponibilizou uma porta serial RS232 no display Toledo 8540 que publica, em uma taxa de 4 a 5 vezes por segundo, o valor medido no display.

[Faça download da documentação do display Toledo 8540](https://s3-sa-east-1.amazonaws.com/time-raposa/wp-content/uploads/2018/03/02165154/protocolo-toledo-8540.pdf)

O manual pode parecer confuso, mas, com certo esforço e paciência, é possível identificar os principais pontos da comunicação.

Passo 2 – Obtenha um emulador do hardware

Este passo é opcional, mas com certeza ajuda muito durante o desenvolvimento quando você não tem um hardware real à sua disposição para os testes iniciais.

Pesquise se existe um emulador oficial produzido pelo próprio fabricante, ou então se alguma alma caridosa implementou e disponibilizou algum emulador extra-oficial em algum fórum especializado, ou blog.

No caso do display Toledo 8540, após 1 ou 2 horas de pesquisa, não encontrei nenhum emulador para o meu sistema operacional Linux (distribuição Debian). Encontrei alguns links somente para Windows.

Sendo assim, pulei esta etapa e prossegui para o próximo passo.

Passo 3 – Emule uma porta serial com o programa `socat`do Linux

O programa `socat`do Linux permite criar portas seriais virtuais (emuladas). Elas podem ser muito úteis para fazer testes rápidos.

Exemplo de execução de comando:

socat -d -d pty,raw,echo=0 pty,raw,echo=0

Na minha máquina, o comando acima imprimiu a seguinte saída (pode variar para o seu caso):

2018/03/02 14:14:51 socat[27691] N PTY is /dev/pts/29

2018/03/02 14:14:51 socat[27691] N PTY is /dev/pts/30

2018/03/02 14:14:51 socat[27691] N starting data transfer loop with FDs [5,5] and [7,7]

Passo 4 – Explore a comunicação serial utilizando o programa `screen`do Linux

O programa `screen` do Linux, entre outras coisas, é capaz de ler continuamente de uma porta serial e imprimir no terminal tudo o que está sendo lido.

É uma ótima ferramenta exploratória.

Por exemplo, para conectarmos o screen às portas virtuais criadas no passo anterior, executamos em uma janela do terminal o comando a seguir:

screen /dev/pts/29

Em outra janela do terminal, conectamos à segunda porta (ver número da porta conforme saída do comando `socat`)

screen /dev/pts/30

Agora, abra as janelas 1 e 2 do terminal lado a lado. Clique na janela 1, e digite alguns caracteres. Surpresa! Os caracteres digitados serão imprimidos na janela 2.

Isso acontece porque o programa `socat` simulou uma conexão serial RS232. Ao digitar caracteres em um lado da conexão, você na verdade está enviando um “sinal elétrico” para o outro lado do cabo.

Passo 5 – Programa em Python de leitura serial

Após algumas análises exploratórias do seu hardware e da comunicação serial, você está pronto para criar um programa em Python para fazer a leitura.

Eu utilizei o Python 2.7, mas provavelmente você não terá dificuldades em migrar esse script para Python 3.

# Arquivo config.py

# Serial port (COM port)

serial\_port = '/dev/ttyUSB0'

# Webservice endpoint

webservice = "https://webservice.mydomain.com/publicar-dados-balanca"

# Webservice authentication token (see weighing\_scale class)

webservice\_token = "4ff86e02514dd8f3610b83ee6947ccf9"

# Name of the scale

weighing\_scale\_identifier = "MINHA\_BALANCA\_1"

# Unit of the scale

unit = 'kg'

# Interval, in seconds, for updating data into the webservice

webservice\_publication\_interval = 1

# Arquivo reader.py

import serial

import sys

import config

while True:

with serial.Serial(

config.serial\_port,

bytesize=serial.SEVENBITS,

parity=serial.PARITY\_EVEN,

stopbits=serial.STOPBITS\_TWO,

baudrate=4800,

timeout=1

) as ser:

line = ser.read(18)

formattedLine = ":".join("{:02x}".format(ord(c)) for c in line)

print formattedLine

sys.stdout.flush()

O que o programa acima faz é muito simples, mas já é um grande passo rumo ao nosso objetivo.

O programa simplesmente define um loop infinito, lendo tudo que chega por meio da porta /dev/ttyUSB0, convertendo para hexadecimal, e imprimindo no terminal, permitindo monitorar com mais qualidade tudo que o hardware (ou emulador) enviou via porta serial, já que nem sempre serão enviados somente dados correspondentes às letras da tabela ASCII.

Agora podemos testar o nosso programa utilizando a conexão serial criada com o comando `socat` anteriormente. Para isso, temos que fazer as seguintes modificações:  
1) Alterar a porta “/dev/ttyUSB0″ para a porta virtual. No meu caso, “/dev/pts/29″.  
2) Importante! Precisamos comentar os parâmetros byteserial=… e parity=… do script acima, pois esses dois parâmetros não funcionam com a porta serial emulada. Entretanto, quando formos utilizar o hardware real, devemos reabilitar esses 2 parâmetros, respeitando as especificações do hardware.

Com o programa do passo anterior, podemos gravar, por alguns minutos, a leitura real proveniente do hardware. Verifique se a porta “/dev/ttyUSB0″ realmente se refere ao hardware do seu display Toledo 8540, e em seguida rode o programa usando o python, redirecionando o output para um arquivo de texto.

python reader.py > serial\_output

Deixe o programa rodando por alguns minutos. O arquivo /tmp/serial\_output será util para você testar o restante do programa sem depender do hardware real. O que estamos fazendo é gravar um arquivo de “replay”, assim podemos emular fielmente o hardware real, sempre que precisarmos.

Se você não tem um hardware real disponível para gravar esse arquivo de “replay”, [baixe aqui o arquivo serial\_output.txt que eu gerei com o meu hardware.](https://s3-sa-east-1.amazonaws.com/time-raposa/wp-content/uploads/2018/03/02165154/serial_output.txt)

Passo 7 – Programa em python para traduzir a saída hexadecimal (gerada pelo programa anterior)

Agora que já temos um arquivo de replay (/tmp/serial\_output), podemos emular mais fielmente os dados enviados pelo hardware. Usaremos esse arquivo para testar o restante do nosso programa.

O próximo programa que criaremos será responsável por traduzir a saída hexadecimal para uma saída legível para humanos.

# Arquivo processor.py

import sys

import time

import config

def validate(byteArray):

if len(byteArray) != 18:

return False

if byteArray[0] != '02':

return False

if byteArray[16] != '0d':

return False

return True

def weight(byteArray):

weightString = ""

for i in range(5, 10):

weightString += intValue(byteArray[i])

return weightString

def intValue(byteString):

return chr(int(byteString, 16))

def parse(line):

bytes = line.split(":")

if (validate(bytes)):

return weight(bytes)

return ""

while 1:

line = sys.stdin.readline()

formattedWeight = parse(line)

if (formattedWeight):

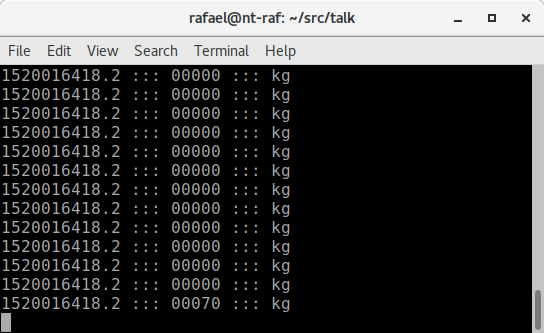
print "%s ::: %s ::: %s" % (time.time(), formattedWeight, config.unit)

sys.stdout.flush()

O programa acima lê, linha a linha, o output gerado pelo programa 1 (reader.py), e converte para o formato desejado. Além disso, o método validate() confere que o primeiro byte da linha é 02, e que o byte da posição 17 é 0d. Esses dois bytes estão especificados na documentação obtida no passo 1, e nos permitem descartar qualquer dado com ruído que possa ter sido recebido pela conexão serial.

Para testar esse programa, vamos imprimir o conteúdo do arquivo de replay de dados (/tmp/serial\_output) e fazer um pipe para o programa tradutor.

cat /tmp/serial\_output | python processor.py

Ao rodar o comando acima, você deverá visualizar a seguinte saída em seu terminal:  


Para rodar junto ao harware real, o comando é similar:

python reader.py | python processor.py

Passo 8 – Publicar os dados traduzidos via webservice

Este passo é opcional, e vai depender da necessidade do seu cliente. No meu caso, optei por publicar via webservice, a cada 1 segundo, o valor lido da balança.

# Arquivo publisher.py

import sys

import datetime

import pprint

import time

import re

from subprocess import Popen

import urllib

import config

LAST\_PUBLICATION\_TIMESTAMP = time.time()

def parse(line):

data = line.strip().split(" ::: ")

if (validate(data)):

return sanitize(data)

def validate(data):

if len(data) != 3:

return False

return True

def sanitize(data):

data[2] = re.sub("[^a-zA-Z]", "", data[2])

return data

def needsPublishing():

interval = time.time() - LAST\_PUBLICATION\_TIMESTAMP

return (interval > config.webservice\_publication\_interval)

def publish(data):

global LAST\_PUBLICATION\_TIMESTAMP

if not needsPublishing():

return True

LAST\_PUBLICATION\_TIMESTAMP = time.time()

data = {

'token': config.webservice\_token,

'weighing\_scale\_identifier': config.weighing\_scale\_identifier,

'timestamp': data[0],

'weight': data[1],

'unit': data[2]

}

cmd = ["curl", "-X", "POST", "--max-time", "10", "--data", urllib.urlencode(data), config.webservice]

try:

p = Popen(cmd)

except Exception as e:

print(e)

pass

while 1:

line = sys.stdin.readline()

data = parse(line)

if data:

pprint.pprint(data)

publish(data)

Para testar o último programa, podemos usar os dados de replay, ou então o hardware real.

Opção 1 – Arquivo de replay

cat /tmp/serial\_output | python processor.py | python publisher.py

Para rodar junto ao harware real, o comando é similar:

python reader.py | python processor.py | python publisher.py

Se tudo deu certo, você terá uma leitura contínua (com taxa de 1 publicação por segundo) sendo publicada em seu webservice.

Passo 9 – Testar software para o Raspberry PI

Felizmente, o Raspberry PI utiliza a distribuição Raspbian, baseada no Debian. Portanto, não precisei fazer nenhum tipo de adaptação para que o meu programa que estava rodando no meu notebook com Debian 9 funcionasse sem problemas no Raspberry PI.

Tive apenas que instalar algumas dependências que não são instaladas por padrão, como por exemplo o python. Mas nada que alguns apt-get install não resolvessem rapidamente.

Para conectar o display Toledo 8540 no Raspberry PI, utilizei um cabo conversor RS232/USB da marca Digitus, com chipset FT232RL. Enfim, qualquer conversor comprado no Mercado Livre deve funcionar bem também.

Passo 10 – Configurar programa para rodar sempre que o Raspberry for conectado à tomada

Essa etapa também é muito simples, já que o próprio Linux possui esse recurso.

Precisamos apenas editar o arquivo /etc/rc.local e incluir o comando que execute os nossos programas. No meu caso, o arquivo /etc/rc.local ficou assim:

#!/bin/sh -e

#

# rc.local

#

# This script is executed at the end of each multiuser runlevel.

# Make sure that the script will "exit 0" on success or any other

# value on error.

#

# In order to enable or disable this script just change the execution

# bits.

#

# By default this script does nothing.

# Print the IP address

\_IP=$(hostname -I) || true

if [ "$\_IP" ]; then

printf "My IP address is %s\n" "$\_IP"

fi

# Start SSH

service ssh start

# Start Balança

cd /home/pi/balanca/ && (python reader.py | python processor.py | python publisher.py) >> /dev/null 2>&1 &

exit 0

Sendo que as únicas alterações que fiz nesse arquivo foram inicializar o servidor SSH (que antes eu instalei com apt-get install), e adicionei a linha que executa o nosso programa em python.

Passo 11 – Instalação do watchdog

Nosso projeto já está funcionando muito bem, mas podemos dar mais robustez ao nosso sistema instalando e configurando o programa `watchdog` do Linux. Não detalherei neste artigo como fazer isso, mas recomendo que você tente estudar e configurar esse programa por sua própria conta.

Com ele, podemos definir alguns critérios de saúde do sistema operacional do nosso Raspberry PI. Quando qualquer desses critérios não estiver sendo respeitado, o aplicativo irá automaticamente rebootar a placa. Assim, não precisamos nos preocupar (tanto!) com desperdício de memória RAM, perda de conexão de rede, CPU load altíssimo, etc.

Dica bônus – Armazene seus dados no banco NoSQL AWS DynamoDB

Você deve ter percebido que, 1 vez por segundo, toda nova leitura da balança é publicada no webservice, mesmo quando o peso é 0kg. Isso pode parecer desperdício à primeira vista, mas é muito útil pois nos passa uma garantia maior de que o sistema está funcionando a todo instante, mesmo quando não há peso na balança.

Mesmo com essa importante vantagem, não podemos ignorar o fato de que essa escolha acarreta um alto custo de armazenamento. Sendo assim, aproveito aqui para oferecer uma dica adicional: o sistema AWS DynamoDB permite armazenar até 25GB de dados gratuitamente, o que estimei que será suficiente para gravar um registro por segundo por mais de 8 anos.

Sendo assim, optei por configurar o webservice para armazenar os meus dados no DynamoDB, e recomendo que você também estude essa opção. Para mais informações, acesse https://aws.amazon.com/pt/dynamodb/.

Até a próxima!

Pronto, se você chegou até aqui, tem um projeto totalmente funcional que faz a leitura contínua do valor medido pelo display Toledo 8540.  
Em caso de dúvidas, me avise pelos comentários!

Até mais.