

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Sistemas Distribuídos
Relatório do Trabalho Prático I

Bruno Lacerda Pêgo - 13.2.8300
Leonardo de Souza Nogueira - 13.2.8289

Professor - Theo Lins

João Monlevade
22 de maio de 2016

Sumário

1	Introdução	2
1.1	Descrição do Problema	2
1.2	Visão geral sobre o funcionamento do programa	2
2	Implementação	3
3	Listagem de testes executados	8
4	Conclusão	13
	Referências	13

1 Introdução

1.1 Descrição do Problema

Neste trabalho prático, deve-se implementar¹ a comunicação distribuída de uma Indústria.

1. Clientes:

- Responsável por solicitar os produtos a Indústria.

2. Fornecedores:

- Responsável por fornecer matéria prima.

3. Controlador de Produção (Multithread):

- Responsável por gerenciar a produção, receber solicitações dos clientes, verificar matéria prima.

4. Linhas de Produções:

- Equipamento de Produção: Responsável por transformar a matéria prima em produto final.
- Sensor Inicial: Verifica os produtos que entraram na linha de produção.
- Sensor Final: Verifica se o produto ficou correto.

1.2 Visão geral sobre o funcionamento do programa

No trabalho temos basicamente quatro módulos onde cada um se relaciona com um ou mais módulos do sistema se necessário.

No nosso caso, o Cliente relaciona com o Controlador de Produção onde ele pode solicitar a produção de cadeiras².

O Controlador se comunica com o cliente, solicita matéria prima para o fornecedor e envia os produtos para produção na Linha de Produção.

O Fornecedor comunica com o Controlador de Produção, enviando quando necessário a matéria prima solicitada.

A Linha de Produção tem como função produzir o produto solicitado pelo Controlador de Produção, a capacidade da linha de produção é de 30 cadeiras no máximo por produção, caso haja falha na produção a mesma deve informar ao Controlador, que deve enviar novamente para a Linha de Produção nova matéria prima para que o total de produtos perfeitos solicitados pelo cliente sejam produzidos.

¹Este trabalho foi implementado utilizando a linguagem *Python3.4*

²A indústria implementada no trabalho produz cadeiras

2 Implementação

No trabalho não foi utilizada alguma estrutura de dados em específico. Durante a execução, pode-se deparar com "MP" que significa *matéria prima*.

1. Cliente:

- Método *main* (os comentários apresentam informações suficiente):

```
1  #Método principal onde se chama o construtor do  
   cliente  
2  def main():  
3      print("Fábrica")  
4      cli = cliente()  
5  
6      #Laco em que o usuário terá a opção de escolher se  
       deseja fazer um pedido  
7      while True:  
8          resp = input("Deseja fazer um pedido [s/n]? ")  
9  
10         #Se o usuário digita 's' ou 'S' então ele  
            digita uma quantidade de produtos para  
            fazer  
11         if resp.lower() == 's':  
12             resp = input("Quantos cadeiras voce deseja  
                            que sejam produzidas? ")  
13             cli.pedido(resp)  
14  
15         # Se o usuário digita 'n' ou 'N' então ele sai  
            da aplicação e a conexão é fechada  
16         # Caso o primeiro comando seja 'n' ou 'N', não  
            houve criação do socket, logo deve-se  
            tratar  
17         # e sair sem tentar fechar socket  
18         elif resp.lower() == 'n':  
19             try:  
20                 cli.s.close()  
21             except Exception:  
22                 pass  
23             break
```

- Método *pedido* (os comentários apresentam informações suficiente):

```
1  #Método onde se cria um socket para comunicação com o  
   controlador de produção  
2  def pedido(self, resp, v = None):  
3  
4      #O bloco abaixo cria o socket, conecta ao  
       controlador e envia a quantidade de  
       produtos que o usuário digitou  
5      self.s = socket.socket()  
6      self.s.connect((self.hostconnectControlador ,  
                       self.port))
```

```

7         self.s.send(resp.encode('utf-8'))
8         #A linha abaixo fica esperando a resposta do
           controlador e logo após fecha o socket de
           comunicação
9         msg = self.s.recv(128).decode('utf-8')
10        print("Resposta do Controlador de Produção: "
              + msg)
11        self.s.close()

```

2. Controlador de Produção:

- Método *tratarPedido* (os comentários apresentam informações suficiente):

```

1  # Abaixo o método de tratar pedido recebido pelo
    cliente e direcionar se precisa de mais matéria
    prima
2      # se precisa de reenviar para linha de produção
    matéria prima para produção de outro produto (
    substituir defeituoso)
3      def tratarPedido(self, cliente, endr):
4          global estoque
5          print("Tem " + str(estoque) + " unidades de MP
              em estoque.")
6
7          # Abaixo Fica à espera de uma mensagem do
    cliente, se a msg não conter nada então
    fecha a conexão
8          msg = cliente.recv(128).decode('utf-8')
9          if not msg:
10             cliente.close()
11          print("Cliente: " + str(endr), "-> Produção
              solicitada: " + msg + " Cadeiras.")
12
13
14          # Abaixo é feita a converção da msg recebida em
    um número INT
15          qtd_solicitada = int(msg)
16
17          # Se o tanto de matéria prima para fazer um
    produto for menor que a quantidade
    solicitada pelo cliente
18          # então é chamada a função de solicitar mais
    matérias primas ao fornecedor, logo após é
    feita a subtração
19          # no estoque a quantidade referente a solicita
    ção do cliente
20          if(estoque < qtd_solicitada):
21              print(str(estoque) + " unidades de MP em
                  estoque...")
22              print("Solicitando matéria prima...")
23              self.solicitarMateriaPrima(str(
                  qtd_solicitada - estoque))

```

```

24         print("Matéria prima recebida...")
25         estoque -= qtd_solicitada
26
27         # A linha de producao tem uma capacidade que é
28         definida pela variável "
29         capacidade_linhaProducao"
30         # que é solicitado a cada iteração do laço e
31         caso necessário é feito no final uma
32         solicitação do restante
33         # Abaixo também é medido o tempo gasto pela
34         linha de produção para fazer os produtos
35         tempo_inicial = time.time()
36         while qtd_solicitada != qtd_solicitada % self.
37             capacidade_linhaProducao:
38                 print("Em produção: " + str(self.
39                     capacidade_linhaProducao))
40                 self.enviarParaProducao(str(self.
41                     capacidade_linhaProducao))
42                 qtd_solicitada -= self.
43                 capacidade_linhaProducao
44
45         if(qtd_solicitada > 0):
46             print("Em produção: " + str(qtd_solicitada
47                 ))
48             self.enviarParaProducao(qtd_solicitada)
49             print("O tempo de produção foi: %s segundos"
50                 %(time.time() - tempo_inicial))
51
52         # Como todos produtos já estão prontos agora é
53         só chamar o método de enviar a resposta
54         para o cliente
55         print("Todos os produtos solicitados estão
56             prontos!")
57         print("Estoque atual: " + str(estoque) + "
58             unidades de MP em estoque.")
59         self.enviarPedidoParaCliente(cliente)
60
61         # Agora a conexão pode ser fechada
62         cliente.close()

```

- Método *solicitarMateriaPrima*:

Assim como nos métodos anteriores, nessa função é criado um *socket* que comunica com o módulo Fornecedor

```

1  # Método de solicitar matéria prima ao fornecedor
2      def solicitarMateriaPrima(self, qtd):
3
4          qtd = str(qtd)
5
6          # Usa-se a variável de estoque global que
7          receberá a matéria prima do fornecedor
8          global estoque

```

```

8         sm = socket.socket()
9         sm.connect((self.hostFornecedor, self.
            porta_fornecedor))
10        sm.send(qtd.encode('utf-8'))
11        materia_prima_recebida = sm.recv(128).decode('
            utf-8')
12        estoque += int(materia_prima_recebida)
13        sm.close()
14        return 0

```

- Método *enviarParaProducao* (os comentários apresentam informações suficientes):

```

1  # Método que envia para a linha de produção a matéria
   prima (nesse caso quantidade de matéria prima)
2      def enviarParaProducao(self, qtd):
3
4          qtd = str(qtd)
5          # Usa-se a variável referente ao estoque para
           controle ao retirar materias primas dele
6          global estoque
7
8          #é criado então um socket para tais transações
9          lp = socket.socket()
10         lp.connect((self.hostLinhaProducao, self.
            porta_linhaProducao))
11         lp.send(qtd.encode('utf-8'))
12
13         # A resposta é a quantidade de materiais com
           defeito que deverão ser reenviados para a
           Linha de Produção
14         resposta = lp.recv(128).decode('utf-8')
15
16         # Enquanto a resposta é diferente de zero
           significa que há necessidade de pegar maté
           ria prima no
17         # estoque e enviar a linha de produção para
           repor os produtos defeituosos. Caso contrá
           rio significa
18         # que não houve perda de produtos ou defeitos
           na linha de produção, então fecha a conexão
19         while resposta != '0':
20
21             if(estoque < int(resposta)):
22                 print("Houve um erro na produção...")
23                 print("Estoque insuficiente -
                    Solicitando materiais...")
24                 self.solicitarMateriaPrima(resposta)
25                 lp.send(resposta.encode('utf-8'))
26                 estoque -= int(resposta)
27             else:
28                 print("Houve um erro na produção...")

```

```

29         lp.send(resposta.encode('utf-8'))
30         estoque -= int(resposta)
31
32         resposta = lp.recv(128).decode('utf-8')
33
34     print("Produção concluída!")
35     lp.close()

```

3. Fornecedor:

- Método *enviarParaProducao* (os comentários apresentam informações suficientes):

```

1  # Método que recebe a quantidade de matéria prima deve
   ser enviada e as retornam
2  def fornecerMateriaPrima(self, cliente, endr):
3      msg = cliente.recv(128).decode('utf-8')
4      qtd_solicitada = int(msg)
5      print(msg + " unidades de MP solicitadas...")
6      print("Enviando " + str(qtd_solicitada) + "
       unidades de MP.")
7      cliente.send(str(qtd_solicitada).encode('utf-8
        '))
8      cliente.close()

```

4. Linha de Produção:

- Método *enviarParaProducao* (os comentários apresentam informações suficientes):

```

1  # Método que fará o produto (recebe a quantidade de
   produto que deve ser feito)
2  def produz(self, qtd):
3      nqtd = int(qtd)
4
5      # Inicialmente define uma variável i como
   contador para o laço e outra
6      # que vai contar o número de produtos
   defeituosos
7      # o laço trata em cada if se o produto está
   defeituoso e caso sim adiciona +1 em
   produtosComDefeitos
8      # Faz um produto de cada vez
9      i = 0
10     produtosComDefeitos = 0
11     while i < nqtd:
12         if self.sensorInicial() == True:
13             if self.equipamentoDeProducao() ==
               True:
14                 if self.sensorFinal() == True:

```



```

15         print("Produto produzido: ", i
16               +1)
17     else:
18         produtosComDefeitos += 1
19     else:
20         produtosComDefeitos += 1
21     else:
22         produtosComDefeitos += 1
23
24     i += 1
25     # Finalmente retorna o resultado de produtos
        com defeito e a quantidade produzida
        independente do defeito
    self.retornaResultado(produtosComDefeitos,
                          nqtd)

```

- Métodos *sensores*:

Os métodos abaixo define a quantide de erro que pode ocorrer na produ-
ção das cadeiras, e pode ter uma taxa de defeito é 10% em cada sensor.

```

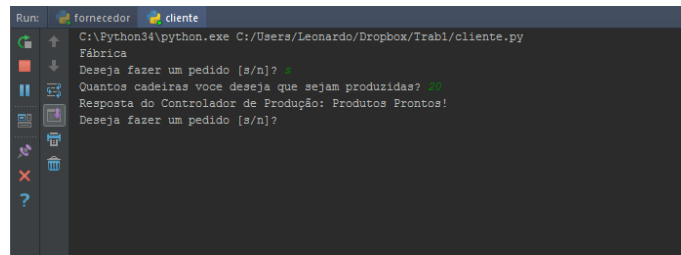
1  # Método que "verifica" se a matéria prima está ou não
   com defeito (apenas gera um número aleatório)
2  def sensorInicial(self):
3      defeito = randrange(0, 10)
4      if defeito == 1:
5          return False
6      return True
7
8  # Método que "verifica" o produto durante o
   processo de produção (apenas gera um número
   aleatório)
9  def equipamentoDeProducao(self):
10     defeito = randrange(0, 10)
11     if defeito == 2:
12         return False
13     return True
14
15     # Método que "verifica" o produto final (apenas
        gera um número aleatório)
16     def sensorFinal(self):
17         defeito = randrange(0, 10)
18         if defeito == 2:
19             return False
20         return True

```

3 Listagem de testes executados

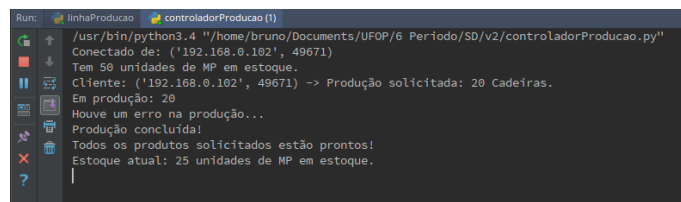
1. Teste 1:

Neste primeiro teste foi solicitado para produzir 20 cadeiras, como pode-se observar pelas imagens houve cadeiras produzidas com defeito, no caso 5 cadeiras defeituosas e com isso foi solicitado novamente a produção de mais 5 cadeiras.



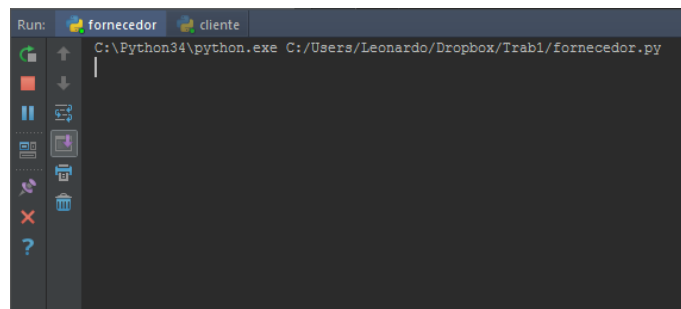
```
Run: fornecedor cliente
C:\Python34\python.exe C:/Users/Leonardo/Dropbox/Trab1/cliente.py
Fábrica
Deseja fazer um pedido [s/n]?
Quantos cadeiras voce deseja que sejam produzidas?
Resposta do Controlador de Produção: Produtos Prontos!
Deseja fazer um pedido [s/n]?
```

Figura 1: Execução do Cliente



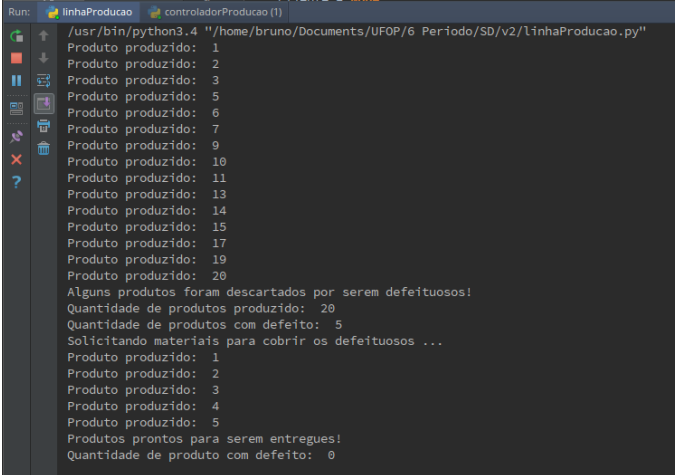
```
Run: linhaProducao controladorProducao (1)
/usr/bin/python3.4 "/home/bruno/Documents/UFOP/6 Período/SD/v2/controladorProducao.py"
Conectado de: ('192.168.0.102', 49671)
Tem 50 unidades de MP em estoque.
Cliente: ('192.168.0.102', 49671) -> Produção solicitada: 20 Cadeiras.
Em produção: 20
Houve um erro na produção...
Produção concluída!
Todos os produtos solicitados estão prontos!
Estoque atual: 25 unidades de MP em estoque.
```

Figura 2: Execução do Controlador de Produção



```
Run: fornecedor cliente
C:\Python34\python.exe C:/Users/Leonardo/Dropbox/Trab1/fornecedor.py
```

Figura 3: Execução do Fornecedor

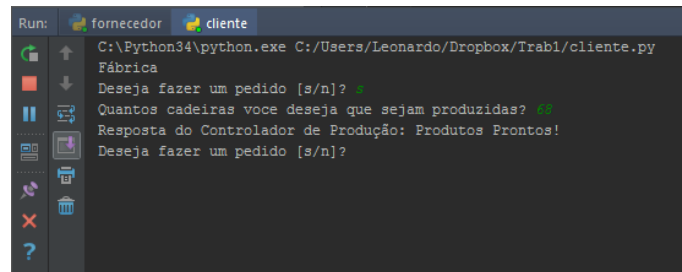


```
Run: linhaProducao | controladorProducao (1)
/usr/bin/python3.4 "/home/bruno/Documents/UFOP/6 Período/SD/v2/linhaProducao.py"
Produto produzido: 1
Produto produzido: 2
Produto produzido: 3
Produto produzido: 5
Produto produzido: 6
Produto produzido: 7
Produto produzido: 9
Produto produzido: 10
Produto produzido: 11
Produto produzido: 13
Produto produzido: 14
Produto produzido: 15
Produto produzido: 17
Produto produzido: 19
Produto produzido: 20
Alguns produtos foram descartados por serem defeituosos!
Quantidade de produtos produzido: 20
Quantidade de produtos com defeito: 5
Solicitando materiais para cobrir os defeituosos ...
Produto produzido: 1
Produto produzido: 2
Produto produzido: 3
Produto produzido: 4
Produto produzido: 5
Produtos prontos para serem entregues!
Quantidade de produto com defeito: 0
```

Figura 4: Execução da Linha de Produção

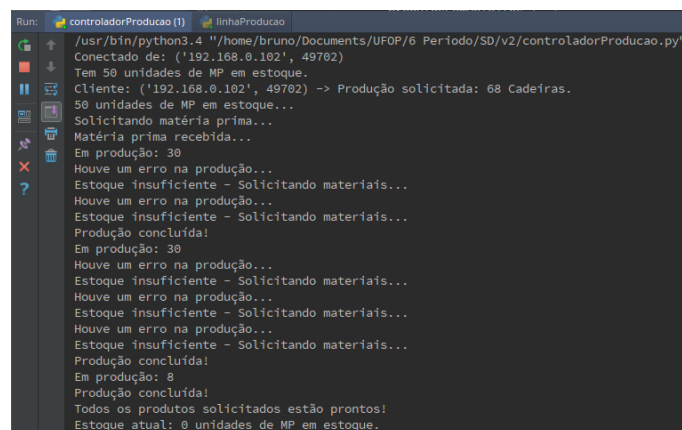
2. Teste 2:

Neste segundo teste foi solicitado para produzir 68 cadeiras, como a linha de produção só produz 30 cadeiras por vez é necessário que o controlador mantenha controle para que haja a produção total das cadeiras solicitadas, pode-se observar pelas imagens que houve cadeiras produzidas com defeito, sempre quando há defeito o controlador solicita novamente que mais cadeiras seja contruídas para atender a solicitação do cliente.



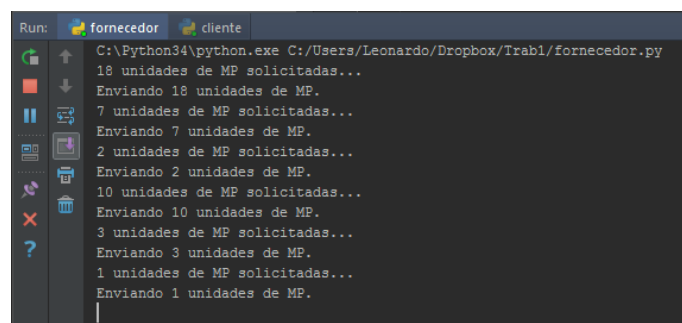
```
Run: fornecedor cliente
C:\Python34\python.exe C:/Users/Leonardo/Dropbox/Trab1/cliente.py
Fábrica
Deseja fazer um pedido [s/n]?
Quantos cadeiras voce deseja que sejam produzidas? 68
Resposta do Controlador de Produção: Produtos Prontos!
Deseja fazer um pedido [s/n]?
```

Figura 5: Execução do Cliente



```
Run: controladorProducao (1) linhaProducao
/usr/bin/python3.4 "/home/bruno/Documents/UFOP/6 Período/SD/v2/controladorProducao.py"
Conectado de: ('192.168.0.102', 49702)
Tem 50 unidades de MP em estoque.
Cliente: ('192.168.0.102', 49702) -> Produção solicitada: 68 Cadeiras.
50 unidades de MP em estoque...
Solicitando matéria prima...
Matéria prima recebida...
Em produção: 30
Houve um erro na produção...
Estoque insuficiente - Solicitando materiais...
Houve um erro na produção...
Estoque insuficiente - Solicitando materiais...
Produção concluída!
Em produção: 30
Houve um erro na produção...
Estoque insuficiente - Solicitando materiais...
Houve um erro na produção...
Estoque insuficiente - Solicitando materiais...
Houve um erro na produção...
Estoque insuficiente - Solicitando materiais...
Produção concluída!
Em produção: 8
Produção concluída!
Todos os produtos solicitados estão prontos!
Estoque atual: 0 unidades de MP em estoque.
```

Figura 6: Execução do Controlador de Produção



```
Run: fornecedor cliente
C:\Python34\python.exe C:/Users/Leonardo/Dropbox/Trab1/fornecedor.py
18 unidades de MP solicitadas...
Enviando 18 unidades de MP.
7 unidades de MP solicitadas...
Enviando 7 unidades de MP.
2 unidades de MP solicitadas...
Enviando 2 unidades de MP.
10 unidades de MP solicitadas...
Enviando 10 unidades de MP.
3 unidades de MP solicitadas...
Enviando 3 unidades de MP.
1 unidades de MP solicitadas...
Enviando 1 unidades de MP.
```

Figura 7: Execução do Fornecedor

```
Run: controladorProducao (1) linhaProducao
/usr/bin/python3.4 "/home/bruno/Documents/UFOP/6 Período/SD/v2/linhaProducao.py"
Produto produzido: 1
Produto produzido: 2
Produto produzido: 5
Produto produzido: 6
Produto produzido: 7
Produto produzido: 8
Produto produzido: 12
Produto produzido: 13
Produto produzido: 14
Produto produzido: 15
Produto produzido: 16
Produto produzido: 17
Produto produzido: 18
Produto produzido: 19
Produto produzido: 20
Produto produzido: 21
Produto produzido: 22
Produto produzido: 23
Produto produzido: 24
Produto produzido: 25
Produto produzido: 26
Produto produzido: 28
Produto produzido: 30
Alguns produtos foram descartados por serem defeituosos!
Quantidade de produtos produzido: 30
Quantidade de produtos com defeito: 7
Solicitando materiais para cobrir os defeituosos ...
Produto produzido: 1
Produto produzido: 2
Produto produzido: 4
Produto produzido: 5
Produto produzido: 6
```

Figura 8: Execução da Linha de Produção (1)

```
Run: controladorProducao (1) linhaProducao
Alguns produtos foram descartados por serem defeituosos!
Quantidade de produtos produzido: 7
Quantidade de produtos com defeito: 2
Solicitando materiais para cobrir os defeituosos ...
Produto produzido: 1
Produto produzido: 2
Produtos prontos para serem entregues!
Quantidade de produto com defeito: 0
Produto produzido: 1
Produto produzido: 2
Produto produzido: 3
Produto produzido: 4
Produto produzido: 7
Produto produzido: 8
Produto produzido: 9
Produto produzido: 10
Produto produzido: 11
Produto produzido: 15
Produto produzido: 16
Produto produzido: 17
Produto produzido: 18
Produto produzido: 19
Produto produzido: 21
Produto produzido: 26
Produto produzido: 27
Produto produzido: 28
Produto produzido: 29
Produto produzido: 30
Alguns produtos foram descartados por serem defeituosos!
Quantidade de produtos produzido: 30
Quantidade de produtos com defeito: 10
Solicitando materiais para cobrir os defeituosos ...
Produto produzido: 1
```

Figura 9: Execução da Linha de Produção (2)

```
Run: controladorProducao (1) linhaProducao
Quantidade de produtos com defeito: 10
Solicitando materiais para cobrir os defeituosos ...
Produto produzido: 1
Produto produzido: 2
Produto produzido: 3
Produto produzido: 4
Produto produzido: 5
Produto produzido: 6
Produto produzido: 10
Alguns produtos foram descartados por serem defeituosos!
Quantidade de produtos produzido: 10
Quantidade de produtos com defeito: 3
Solicitando materiais para cobrir os defeituosos ...
Produto produzido: 1
Produto produzido: 2
Alguns produtos foram descartados por serem defeituosos!
Quantidade de produtos produzido: 3
Quantidade de produtos com defeito: 1
Solicitando materiais para cobrir os defeituosos ...
Produto produzido: 1
Produtos prontos para serem entregues!
Quantidade de produto com defeito: 0
Produto produzido: 1
Produto produzido: 2
Produto produzido: 3
Produto produzido: 4
Produto produzido: 5
Produto produzido: 6
Produto produzido: 7
Produto produzido: 8
Produtos prontos para serem entregues!
Quantidade de produto com defeito: 0
```

Figura 10: Execução da Linha de Produção (3)

4 Conclusão

A implementação do trabalho na linguagem de programação Python não foi muito complicada devido esta oferecer uma grande simplicidade de implementação. Contudo, houve um trabalho a mais para conseguir desenvolver a comunicação entre os módulos do sistema de modo que ficassem sincronizados e coerentes.

Referências

- [1] Socket basico. <http://wiki.python.org.br/SocketBasico>. Acessado em: 17-05-2016.
- [2] Socket objects. <https://docs.python.org/2/library/socket.html#socket-objects>. Acessado em: 17-05-2016.
- [3] Socket programming howto. <https://docs.python.org/2/howto/sockets.html>. Acessado em: 15-05-2016.
- [4] socket — low-level networking interface. <https://docs.python.org/3/library/socket.html#>. Acessado em: 20-05-2016.