Documentação do Trabalho prático 3

Nome: Alessandro Azevedo Duarte
Matrícula: 2017072642

Irei comentar na documentação sobre os pontos do código os quais alterei em relação ao último trabalho entregue.

Para entender melhor a parte de comunicação TCP/IP em python utilizei desse tutorial:

https://steelkiwi.com/blog/working-tcp-sockets/

E tive alguns problemas com relação ao envio e recebimento dos dados e como na maioria dos casos o stack Overflow nos ajuda muito nesses momentos, e nesse momento essa issue me ajudou a resolver o problema:

https://stackoverflow.com/questions/42612002/python-sockets-error-typeerror-a-bytes-like-object-is-required-not-str-with

Optei por fazer a comunicação TCP/IP no mesmo arquivo e utilizei do WSL (Windows Subsystem for Linux) para realizar os meus testes e não tive problemas de EOF como relatado por alguns colegas na aula de discussão sobre o trabalho.

Para aplicar os conceitos de processos pai e filho utilizei da biblioteca de multiprocessing como exemplificado na aula ATR - aula 34: IPC - Memória compartilhada e como demonstrado essa parte que cria os processos e imprime os gráficos é a main do programa.

```
if __name__ == '__main__':
    #Criando as threads
    print("Simulação de um controle de 2 tanques de água")
    inicio = time.time()
    interface_thread = threading.Thread(target = interface,name = 'Interface')
    process_thread_1 = threading.Thread(target = tan1,name = 'Tanque nº 1',args=(coef1,r1Min,r1Max,h1Max))
    process_thread_2 = threading.Thread(target = tan2,name = 'Tanque nº 2',args=(coef2,r2Min,r2Max,h2Max))
    softPLC_thread = threading.Thread(target = controlador,name = 'Controlador')
    logger_thread = threading.Thread(target = logger,name = 'Logger')
    synopitc_process = multiprocessing.Process(target = synopitc,name= 'Synopitc')

    interface_thread.start()
    process_thread_1.start()
    process_thread_2.start()
    synopitc_process.start()
    softPLC_thread.start()
```

Também criei mais uma função que é a responsável pelo sinóptico

```python
def synopitc():

    #Listen
    listen = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    listen.bind( ('127.0.0.1', 5032))
    listen.listen ( 0 )
    print('\nEsperando o Cliente \n')
    fid = open('historiador.txt','w')
    fid.write('Alturas e saidas\n')
    while True:
        cliente, endereco = listen.accept()
        data = cliente.recv(1024)
        print('\nO cliente  é: \n')
        print(endereco)
        print('\nO dado  é: \n')
        x = data.decode('utf-8')
        print(x)
        aux = time.gmtime(time.time() - inicio)
        auxResult = time.strftime("%H:%M:%S",aux)
        fid = open('historiador.txt','a+')
        fid.write('No Tempo de execucacao= ' + str(auxResult) +' | ' + x + '\n\n')
        time.sleep(1)
        if not x:
            exit
        cliente.sendall(data)
    cliente.close()
    print("O processo do Sinoptico finalizou")
```

Ela também representa o servidor da conexão TCP onde através da utilização da biblioteca de sockets definimos um ip e uma porta onde ficará escutando, esperando por uma conexão de cliente e junto desse processo o historiador é criado para escrita dos dados.

Temos um loop para que essa conexão sempre aconteça e a função de accept recebe dois dados que são o cliente e o endereço desse cliente que conecta no servidor.

Como explicado no começo tive alguns problemas relacionados ao envio dos dados e como sugerido na issue do stackoverflow utilizei das funções encode para enviar os dados e da função de decode para receber os mesmos, sendo que nas duas foi utilizado o padrão utf-8, após essa etapa gerei um timestamp, semelhante ao utilizado na função do logger para que os dados armazenados no historiador tenha uma referência temporal.

E da mesma forma como feito no logger utilizei a função open com a tag de ‘a+’ para escrever os dados no arquivo. Temos posteriormente uma condição que se não forem enviados dados a função tem uma saída através do exit, e como demonstrado em alguns pontos relacionados ao protocolo TCP temos uma confirmação que é enviada posteriormente, e após esse processo temos um fechamento da conexão e finalização do processo.

Optei por colocar o cliente da minha conexão dentro do controlador, como segue o código.

```
    listen = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    listen.connect( ('127.0.0.1', 5032))
    output0 = str(alt1)
    output1 = str(alt2)
    output2 = str(exit1)
    output3 = str(exit2)
    output4 = str(' A altura do tanque 1 e: ' + output0 + ' | A altura do tanque 2 e: ' + output1 + '\nA vazao dc
    listen.send(output4)
    data = listen.recv(1024)
    listen.close()
    if not data:
        exit
    print("Received\n")
    repr(data)
```

Da mesma forma o socket é criado e uma conexão é realizada no ip e porta definidos.

A minha estratégia para enviar os dados necessários foi passar os mesmos para uma string e depois concatenar todos eles, para que eu fizesse o envio uma única vez com todos os dados, através da função send. Como relatado anteriormente temos um recv no cliente devido a confirmação dos dados, onde após o recebimento uma mensagem de received é impressa na tela.

```
Simulação de um controle de 2 tanques de água

Se deseja alterar o setpoint do tanque 1, digite o valor e tecle enter


Se não digite 0 e tecle enter


O programa continua rodando e você pode acompanhar pelo arquivo log.txt

-->
Esperando o Cliente


O cliente  é:

('127.0.0.1', 52649)

O dado  é:

 A altura do tanque 1 e: 0.0 | A altura do tanque 2 e: 0.0
A vazao do tanque 1 e: 0.0 | A vazao do tanque 2 e: 0.0
Received
```

Esse é o resultado da tela após o início do programa e dessa forma o usuário pode digitar o valor de referência para os setpoints dos tanques a qualquer momento.

Quando o usuário digita algum valor, uma mensagem maior separa os dados para que se for necessário o mesmo pode voltar um pouco as mensagens e verificar o que foi digitado.

```
 A altura do tanque 1 e: 0.12753165952078582 | A altura do tanque 2 e: 0.2990000154497497
A vazao do tanque 1 e: 0.13680924471536474 | A vazao do tanque 2 e: 0.18499921039593809




 ********************************VALOR DIGITADO PARA O SETPOINT 1**********************




3




********************************************************************************

Se deseja alterar o setpoint do tanque 2, digite o valor e tecle enter
```

E da mesma forma o setpoint é alterado e esses dados podem ser acompanhados no log e também no historiador.

Segue um print do arquivo de log e do historiador, os dois são criados automaticamente pelo programa.

tp1p3.py | historiador.txt | readme.txt

historiador.txt

```
1   Alturas e saidas
2   No Tempo de execucacao= 00:00:02 |  A altura do tanque 1 e: 0.0 | A altura do tanque 2 e: 0.0
3   A vazao do tanque 1 e: 0.0 | A vazao do tanque 2 e: 0.0
4
5   No Tempo de execucacao= 00:00:03 |  A altura do tanque 1 e: 0.019613945760210284 | A altura do tanque 2 e: 0.046220
6   A vazao do tanque 1 e: 0.04009756137437332 | A vazao do tanque 2 e: 0.053594264911923775
7
8   No Tempo de execucacao= 00:00:04 |  A altura do tanque 1 e: 0.12757997654972025 | A altura do tanque 2 e: 0.2993620
9   A vazao do tanque 1 e: 0.13683311400243045 | A vazao do tanque 2 e: 0.18511881393280738
10
11  No Tempo de execucacao= 00:00:05 |  A altura do tanque 1 e: 0.2488950568855149 | A altura do tanque 2 e: 0.52235779
12  A vazao do tanque 1 e: 0.19500179038783655 | A vazao do tanque 2 e: 0.24867886993357322
```

tp1p3.py | log.txt | readme.txt

log.txt

```
1   Alturas e saidas
2   Tempo de exucacao= 00:00:00 |  altura 1= 0 |  altura 2= 0 |  saida 1= 0 |  saida 2= 0 |
3   Tempo de exucacao= 00:00:01 |  altura 1= 0 |  altura 2= 0 |  saida 1= 0 |  saida 2= 0 |
4   Tempo de exucacao= 00:00:02 |  altura 1= 0.0 |  altura 2= 0.0 |  saida 1= 0.0 |  saida 2= 0.0 |
5   Tempo de exucacao= 00:00:03 |  altura 1= 0.0 |  altura 2= 0.0 |  saida 1= 0.0 |  saida 2= 0.0 |
6   Tempo de exucacao= 00:00:04 |  altura 1= 0.08995869316783002 |  altura 2= 0.2187019336125663 |  saida 1= 0.11427836
7   Tempo de exucacao= 00:00:05 |  altura 1= 0.18818219029418654 |  altura 2= 0.41392800890889553 |  saida 1= 0.1689425
8   Tempo de exucacao= 00:00:06 |  altura 1= 0.29257351953614114 |  altura 2= 0.591108291561179 |  saida 1= 0.212352600
9   Tempo de exucacao= 00:00:07 |  altura 1= 0.40099739606418133 |  altura 2= 0.7533423351568355 |  saida 1= 0.24967341
10  Tempo de exucacao= 00:00:08 |  altura 1= 0.5118633911909872 |  altura 2= 0.9023984586736666 |  saida 1= 0.282865022
11  Tempo de exucacao= 00:00:09 |  altura 1= 0.6238004905095671 |  altura 2= 1.039723132921002 |  saida 1= 0.3131911018
12  Tempo de exucacao= 00:00:10 |  altura 1= 0.7380253366325255 |  altura 2= 1.1653759268253017 |  saida 1= 0.34106667
13  Tempo de exucacao= 00:00:11 |  altura 1= 0.8510923430647327 |  altura 2= 1.2811395233035214 |  saida 1= 0.366604992
14  Tempo de exucacao= 00:00:12 |  altura 1= 0.9626768526258643 |  altura 2= 1.388109260943476 |  saida 1= 0.3901903498
```

Foi bastante interessante utilizar a conexão TCP/IP, que são conceitos que também foram aprendidos nas aulas de redes e ver a aplicação dos mesmos em um sistema de controle na prática.

Um ponto bastante interessante é que a comunidade de programação sempre nos ajuda muito a resolver problemas e o stack overflow foi de grande ajuda para desenvolver esse trabalho.

Essa junção dos conceitos de diversas matérias é muito interessante e ter essa experiência na faculdade em conjunto com um estágio é ainda mais legal, pois conseguimos verificar tudo na prática também.

Uma possível melhoria para o trabalho é o desenvolvimento de um sinóptico mais elaborado, onde as mensagens e os dados poderiam ter cores que ajudariam o operador a entender o que está acontecendo no processo.