ESTI – ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO
GRADUAÇÃO EM ANÁLISE E DESENVOLVIMENTO
DE SISTEMAS

PROJETO DE BLOCO

Caio M. Veloso Dias

Rio de Janeiro, 22 de Setembro de 2018.

Sumário

Introdução **3**

Desenvolvimento do projeto **4**

Resultados e Análise **31**

Conclusão **42**

Introdução

O projeto foi pensado de forma a facilitar o usuário a ter informações de memória, cpu, disco, arquivos, diretório, processos e redes de forma clara e de fácil domínio, para usuários com pouca familiaridade com o assunto.

O projeto consiste em um sistema básico escrito em python, baseado em comunicação via socket, cliente/servidor, comunicação essa estabelecida através do protocolo TCP.

Da parte do cliente, responsável por receber a entrada de instruções do usuário, e formado por um menu de dez opções, a cada opção escolhida o cliente informa ao servidor qual informação ele quer e após uma busca feita pelo servidor, essa informação é entregue ao cliente e o mesmo apresenta ao usuário.

Desenvolvimento do projeto

O projeto foi desenvolvido na linguagem python em sua versão 3.6, foram utilizadas as bibliotecas socket, pickle, cpuinfo, psutil e OS para auxiliar o desenvolvimento. Toda a comunicação do projeto foi feita baseada em socket por isso foi dividido em dois arquivos, client e server e foi escolhido o protocolo TCP para a comunicação com a intenção de prevenir perdas de pacotes e uma comunicação mais instável.

Cliente:

Os imports das dependências necessárias para o projeto, estarei disponibilizando a documentação das mesmas

socket, pickle, os, time, psutil

```
import socket
import pickle
import os
import time
import psutil as ps
```

O projeto possui uma única constante, GIGAFACTOR, que é responsável por ter a operação auxiliar para conversão de KB para GB.

```
GIGAFACTOR = float(1 << 30)
```

O operador binário << tem a função de elevar 1 por 30.

Foram externalizadas algumas funcionalidades da parte do cliente em funções devido a simplicidade do código e melhor entendimento do mesmo.

Das funções:

formatCpuInfoPercent()

```
# Formata as informações de percentual de uso da CPU e as apresenta.
def formatCpuInfoPercent(res):
    c = 1

    for i in range(0, len(res)):
        print('\n'+'\t'+"Porcentagem de CPU no nucleo", c, "usada:", res[i], "%")
        c += 1
```

Tem a finalidade de formatar e apresentar o dado recebido do servidor referente a ao percentual de uso da CPU.

Recebe uma propriedade que representa o percentual de uso, e tem uma variável **c** que funciona como contadora para a quantidade de núcleos possuidos pela CPU.

formatCpuInfo()

```
# Formata as informações de CPU e as apresenta.
def formatCpuInfo(info):
    print('\n'+'\t'+"Informações da CPU da sua maquina:")
    print('\t'+"Nome / modelo:", info[0])
    print('\t'+"Tipo de arquitetura:", info[1])
    print('\t'+"Plavra do processador:", info[2])
    print('\t'+"Frequencia total da CPU:", info[3])
    print('\t'+"Frequencia atual da CPU:", info[4])
    print('\t'+"Frequencia minima da CPU:", info[5])
    print('\t'+"Numero de nucleos físicos:", info[6])
    print('\t'+"Numero de nucleos lógicos:", info[7], ''+'\n')
```

Tem a finalidade de formatar e apresentar as informações recebidas do servidor referente

a CPU, nome, arquitetura, número de núcleos, etc.

Recebe uma propriedade que representa uma lista com os dado da CPU já dispostos no

index da lista ordenados de acordo com a disposição das informações.

getFileInformation()

```
# Formata as informações de uma lista de arquivos
def formatFileInformation(path, list_files):

    dic_files = {}

    for i in list_files:
        if os.path.isfile(i):
            dic_files[i] = []
            dic_files[i].append(os.stat(i).st_size)
            dic_files[i].append(os.stat(i).st_atime)
            dic_files[i].append(os.stat(i).st_mtime)

    titulo = '{:11}'.format("Tamanho")  # 10 caracteres + 1 de espaço

    # Concatenar com 25 caracteres + 2 de espaços
    titulo = titulo + '{:27}'.format("Data de Modificação")

    # Concatenar com 25 caracteres + 2 de espaços
    titulo = titulo + '{:27}'.format("Data de Criação")

    titulo = titulo + "Nome"

    print('\n'+'\t'+titulo)

    for i in dic_files:
        kb = dic_files[i][0] / 1000
        tamanho = '{:10}'.format(str('{:.2f}'.format(kb) + ' KB'))
        print('\t'+tamanho, time.ctime(dic_files[i][2]), " ", time.ctime(dic_files[i][1])," ", i)
    print('')
```

Tem a finalidade de formatar e apresentar os dados recebidos do servidor referente a arquivos.

Recebe dois parâmetros ***path*** representando o caminho escolhido pelo usuário e ***list_files*** que representa a lista de arquivos encontrados no servidor com base no caminho que o usuário forneceu.

possui uma variável em seu escopo, ***dic_files*** que representa o dicionario de arquivos que será exibido ao usuário no final.

formatDirInformation()

```
# Formata os dados de um diritorio
def formatDirInformation(path, dir_size, create_date):

    dir_name = os.path.split(path)

    # Formata os dados
    dir_size = '{:16}'.format(str('{:.2f}'.format(dir_size) + ' KB'))
    create_date = '{:29}'.format(time.ctime(create_date))

    #formata titulo
    titulo = '{:17}'.format("Tamanho")  # 13 caracteres + 1 de espaço

    titulo = titulo + '{:30}'.format("Data de Criação")

    titulo = titulo + '{:30}'.format("Nome do diretorio")

    # Mostra informações formatadas
    print('\n'+'\t'+titulo)
    print('\t'+dir_size, create_date, dir_name[1] + '\n')
```

Tem a finalidade de formatar e apresentar os dados recebidos do servidor referente a diretórios.

Recebe três parâmetros, ***path*** representando o caminho do diretório escolhido pelo usuário, ***dir_size*** que representa o tamanho do diretório escolhido pelo usuário e ***create_date*** representando a data de criação do diretório fornecido pelo usuário.

Possui uma variável em seu escopo, ***dir_name*** que receberá o nome do diretório através do retorno do método ***split()*** passando o path escolhido pelo usuário.

formatProcessInformation()

```python
# Formata os dados de um PID
def formatProcessInformation(pid, current_process):
    print('\n'+'\t'+"PID", pid)
    print('\t'+"Nome:", current_process.name())
    print('\t'+"Executável:", current_process.exe())
    print('\t'+"Tempo de criação:", time.ctime(current_process.create_time()))
    print('\t'+"Tempo de usuário:", current_process.cpu_times().user, "s")
    print('\t'+"Tempo de sistema:", current_process.cpu_times().system, "s")
    print('\t'+"Percentual de uso de CPU:", current_process.cpu_percent(interval=1.0), "%")
    perc_mem = '{:.2f}'.format(current_process.memory_percent())
    print('\t'+"Percentual de uso de memória:", perc_mem, "%")

    # RSS: Resident set size e VMS: Virtual Memory Size
    mem = '{:.2f}'.format(current_process.memory_info().rss / 1024 / 1024)
    print('\t'+"Uso de memória:", mem, "MB")
    print('\t'+"Número de threads:", current_process.num_threads(), '' + '\n')
```

Tem a finalidade de formatar e apresentar os dados recebidos do servidor referente a um processo, processo esse capturado através do PID encontrado pelo servidor.

Recebe dois parâmetros, ***pid*** representa o PID do processo escolhido pelo usuário e ***current_process*** que representa o resultado da captura das informações do processo feita pelo servidor.

Possui uma variável em seu escopo, ***perc_mem*** que formata para apresentar o percentual de uso de memória que tem o processo escolhido.

formatNetworkInformation()

```python
# Formata as informações de redes
def formatNetworkInformation(interfaces):
    names = []

    # Obtém os nomes das interfaces primeiro
    for i in interfaces:
        names.append(str(i))

    # Depois, imprimir os valores:
    for i in names:
        print('\n'+i+":")

        for j in interfaces[i]:
            #print("\t"+str(j))
            print("\t"+'Endereço IP: '+str(j.address))
            print("\t"+'Familia: '+str(j.family))
            print("\t"+'Netmask: '+str(j.netmask)+'\n')
```

Tem a finalidade de formatar e apresentar os dados recebidos do servidor referente as interfaces de rede em que a máquina se encontra.

Recebe um único parâmetro, ***interfaces*** que representa a lista de interfaces que o servidor buscou e encontrou.

Possui uma variável em seu escopo, ***names*** ela recebe os nomes das interfaces existentes para melhor formatação das informações e exibição para o usuário.

Foi escolhido uma estrutura de try/except para a comunicação com o servidor e uma estrutura de looping com while para a constância do menu para o usuário.

Da criação do socket:

```
# Cria o socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Como dito, foi estabelecida uma conexão via protocolo TCP, utilizando assim a função socket passando os parâmetros de identificação da comunicação, ***family*** = socket.AF_INET (identifica a família, nesse caso ipv4) e ***type*** = socket.SOCK_STREAM (identifica o tipo de conexão, nesse caso TCP).

```
# Tenta se conectar ao servidor
s.connect((socket.gethostname(), 14562))
```

O método ***connect*** do socket tenta estabelecer uma conexão com o servidor, recebe como parâmetro o ***nome do cliente*** (socket.gethostname()) e uma ***porta*** (nesse caso 14562) para estabelecer a conexão, que precisa ser a mesma que a do server.

Da estrutura de looping:

```
print('\n'+'\t'+'-------    Escolha uma operação.'+'\n')
print('\t'+'-------    1 Percentual de memoria RAM.')
print('\t'+'-------    2 Percentual deu so da CPU.')
print('\t'+'-------    3 Percentual deu so de Disco.')
print('\t'+'-------    4 Informações da CPU.')
print('\t'+'-------    5 IP da maquina.')
print('\t'+'-------    6 Informações de arquivos de um diretorio.')
print('\t'+'-------    7 Informações sobre um diretorio.')
print('\t'+'-------    8 Informações sobre um processo.')
print('\t'+'-------    9 Informações sobre a rede presente.')
print('\t'+'-------    fim para sair.'+'\n')
msg = input('>>> ')
```

Para receber as informações do usuário foi feita uma estrutura de print's mostrando as opções válidas para o usuário e um input para receber a opção escolhida.

```
# Finaliza o lado do cliente
if msg == 'fim':
    s.send(msg.encode('ascii'))
    break
```

Caso o usuário entre com a string ***"fim"*** , será enviado ao servidor a mensagem de fim que encerra a conexão e encerra o looping do while, encerrando assim a interação do usuário com o sistema.

Das opções do menu:

```
# Retorna porcentagem de memoria
if msg == '1':

    # Envia mensagem vazia apenas para indicar a requisição
    s.send(msg.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(1024)

    # Converte os bytes para lista
    res_mem_percent = pickle.loads(bytes)

    print('\n'+'\t'+"Porcentagem de memoria usada:", res_mem_percent[0], "%"+'\n')
```

No caso um, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, operação essa que retorna a porcentagem de uso de memória RAM.

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e será formatada para ser exibida para o usuário.

```
# Retorna porcentagem de uso da CPU
if msg == '2':

    # Envia mensagem vazia apenas para indicar a requisição
    s.send(msg.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(1024)

    # Converte os bytes para lista
    res_cpu_percent = pickle.loads(bytes)

    formatCpuInfoPercent(res_cpu_percent)
    print('')
```

No caso dois, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, operação essa que retorna a porcentagem de uso de CPU atual.

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e é chamada a função ***formatCpuInfoPercent()*** que formata e exibe a informação para o usuário.

```
# Retorna porcentagem de uso de disco
if msg == '3':

    # Envia mensagem vazia apenas para indicar a requisição
    s.send(msg.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(1024)

    # Converte os bytes para lista
    res_disk_percent = pickle.loads(bytes)

    print('\n'+'\t'+"Porcentagem de disco usada:", res_disk_percent[0], "%"+'\n')
    print('')
```

No caso três, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, operação essa que retorna a porcentagem de uso de disco.

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e será formatada para ser exibida para o usuário.

```
# Retorna informações da CPU
if msg == '4':

    # Envia mensagem vazia apenas para indicar a requisição
    s.send(msg.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(1024)

    # Converte os bytes para lista
    res_cpu_info = pickle.loads(bytes)

    formatCpuInfo(res_cpu_info)
```

No caso quatro, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, operação essa que retorna informações da CPU.

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e é chamada a função ***formatCpuInfo()*** que formata e exibe a informação para o usuário.

```
# Retorna o ip da maquina
if msg == '5':

    # Envia mensagem vazia apenas para indicar a requisição
    s.send(msg.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(1024)

    # Converte os bytes para lista
    res_ip = pickle.loads(bytes)

    print('\n'+'\t'+'Ip da maquina:', res_ip[0], ''+'\n')
```

No caso cinco, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, operação essa que retorna o endereço IP da máquina.

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e será formatada para ser exibida para o usuário.

```
# Retorna as informações de um arquivo com base no path.
if msg == '6':
    s.send(msg.encode('ascii'))

    # recebe e envia o path para o servidor
    print('\n'+'\t'+'Por favor, digite o caminho do arquivo.\n')
    path = input('>>> ')
    s.send(path.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(1024)

    # Converte os bytes para lista
    res_files = pickle.loads(bytes)

    formatFileInformation(path, res_files)
```

No caso seis, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, após o servidor identificar, o usuário deverá informar um caminho válido, após a entrada do dado será enviada uma nova mensagem ao servidor que irá retornar as informações referentes aos arquivos contidos em um diretório ou de um arquivo em si.

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e é chamada a função ***formatFileInformation()*** que formata e exibe a informação para o usuário.

```
# Retorna as informações de um diretorio com base no path.
if msg == '7':
    s.send(msg.encode('ascii'))

    # recebe e envia o path para o servidor
    print('\n'+'\t'+'Por favor, digite o caminho de um diretorio.\n')
    path = input('>>> ')
    s.send(path.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(1024)

    # Converte os bytes para lista
    res_files = pickle.loads(bytes)

    formatDirInformation(path, res_files[0], res_files[1])
```

No caso sete, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, após o servidor identificar, o usuário deverá informar o caminho de um diretório válido, após a entrada do dado será enviada uma nova mensagem ao servidor que irá retornar as informações referentes ao diretório informado.

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e é chamada a função ***formatDirInformation()*** que formata e exibe a informação para o usuário.

```python
# Retorna as informações de um processo com base no seu nome.
if msg == '8':
    s.send(msg.encode('ascii'))

    # recebe e envia o path para o servidor
    print('\n'+'\t'+'Por favor, digite o nome do processo.\n')
    path = input('>>> ')
    s.send(path.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(1024)

    # Converte os bytes para lista
    res_process = pickle.loads(bytes)

    formatProcessInformation(res_process[0], res_process[1])
```

No caso oito, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, após o servidor identificar, o usuário deverá informar o nome de de um processo ativo, após a entrada, será enviada uma nova mensagem ao servidor que irá retornar as informações referentes ao processo escolhido.

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e é chamada a função ***formatProcessInformation()*** que formata e exibe a informação para o usuário.

```python
# Retorna as informações da rede em que a maquina se encontra.
if msg == '9':
    s.send(msg.encode('ascii'))

    # Recebe a resposta do servidor.
    bytes = s.recv(2048)

    # Converte os bytes para lista
    res_network = pickle.loads(bytes)

    formatNetworkInformation(res_network[0])
```

No caso nove, será enviado uma mensagem, convertida em bytes, ao servidor indicando qual operação o usuário quer realizar, operação essa que retorna as informações da rede em que a máquina se encontra..

A resposta do servidor será convertida de volta através do pickle, uma vez que é usado no servidor o pickle para enviar a mensagem, e será chamada a função ***formatNetworkInformation()*** que formata e exibe a informação para o usuário.

Com a última operação finaliza as interações do usuário com o sistema. caso aconteça algum erro durante as operações, ele irá cair na exceção e será exibido para o usuário.

```python
except Exception as erro:
    print('\n'+str(erro)+'\n')
```

Ao looping ser encerrado o socket e finalizado e é informado ao usuário.

```python
# Fecha o socket
s.close()

input("Pressione qualquer tecla para sair...")
```

Servidor

Os imports das dependências necessárias para o projeto, estarei disponibilizando a documentação das mesmas

socket, pickle, os, psutil

```
# Servidor
import socket
import pickle
import cpuinfo
import psutil as ps
import os
```

Da criação da conexão e espera de conexão:

```
# Cria o socket
socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Obtem o nome da máquina
host = socket.gethostname()

porta = 14562

# Associa a porta
socket_servidor.bind((host, porta))

# Escutando...
socket_servidor.listen(1)

print("Servidor de nome", host, "esperando conexão na porta", porta)

# Aceita alguma conexão
(cliente, addresse) = socket_servidor.accept()

print("Conectado a:", str(addresse))
```

A variável ***socket_servidor*** cria o socket passando as configurações de família e tipo, como

explicado no cliente, family = socket.AF_INET (ipv4) e type = socket.SOCK_STREAM (TCP).

o nome da máquina é armazenado na variável ***host*** e a porta que será utilizada pelo servidor na variável ***porta,*** logo em seguida é feito o bind da configuração do socket com o método ***socket_servidor.bind()*** passando o nome da máquina e a porta a ser utilizada.

com o método ***socket_servidor.listen()*** inicia-se o processo de escuta do servidor, no qual ele fica escutando uma conexão com um cliente, o último passo é aceitar uma conexão, para isso é utilizado o método ***socket_servidor.accept()***, é retornado dele as informações do cliente que estabeleceu a conexão, ***client*** que representa as informações do cliente e ***adderesse*** que representa o endereço do cliente, ao estabelecer a conexão é informado no terminal o endereço do cliente em questão.

Das funções:

getMemoryUsagePercent()

```
# Porcentagem do uso de memoria
def getMemoryUsagePercent():
    memory_used = ps.virtual_memory().percent
    return memory_used
```

Tem a finalidade de retornar o percentual de uso de memoria da maquina.

é utilizado o método do módulo psutil, ***virtual_memory().percent*** para retornar o percentual de memória usada.

getCpuUsagePercent()

```
# Porcentagem do uso de CPU;
def getCpuUsagePercent(i):
    return ps.cpu_percent(interval=1, percpu=True)[i]
```

Tem a finalidade de retornar o percentual de uso da CPU da máquina, o metodo recebe um numeral ***i*** que é utilizado como contador para os nucleos da CPU, é utilizado o método do módulo psutil, ***cpu_percent()***, passando as propiedades ***interval*** (intervalo de 1 segundo para cada busca) e ***percpu*** (retorna o percentual da informação), para retornar o percentual de cpu.

getCpuInfo()

```
# Informações de CPU
def getCpuInfo(info):
    temp = []
    temp.append(info['brand'])
    temp.append(info['arch'])
    temp.append(info['bits'])
    temp.append(ps.cpu_freq().max)
    temp.append(ps.cpu_freq().current)
    temp.append(ps.cpu_freq().min)
    temp.append(ps.cpu_count(logical=False))
    temp.append(ps.cpu_count() - ps.cpu_count(logical=False))
    return temp
```

Tem a finalidade de retornar uma lista com as principais informações da CPU da máquina, o método recebe um parâmetro ***info*** que representa o dicionário contendo informações da CPU e para capturar o resto das informações é utilizado o método do módulo psutil, ***cpu_freq()*** e ***cpu_count()***, é retornada uma lista temporaria com toda a informação.

getDirSize()

```python
# Tamanho de um diretorio
def getDirSize(path):

    ## calcula o tamano de um diretorio
    if os.path.isdir(path):
        somador = 0

        lista = os.listdir(path)

        for i in lista:
            p = os.path.join(path, i)
            if os.path.isfile(p):
                somador = somador + os.stat(p).st_size
        return somador / 1000
    else:
        return "O diretório", '\'' + path + '\'', "não existe."
```

Tem a finalidade de retornar o tamanho total de um diretório, o método recebe um parâmetro ***path*** que representa o caminho do diretório escolhido pelo usuário.

para uma melhor experiência do usuário, é feita uma verificação se o caminho é realmente um diretório caso contrário é retornado uma mensagem para o cliente, são criadas as variáveis ***somador*** e ***lista*** que representam um somador que receberá o tamanho de tudo que estiver dentro do diretório escolhido e a lista de arquivos e diretórios dentro do path escolhido, por fim é retornado o somador dividido por mill para melhor representatividade da medida KB.

getPIDByName()

```python
# Retorna o pid de um processo com base no nome do mesmo
def getPIDByName(name):

    lp = ps.pids()

    lista_pid = []

    for i in lp:
        p = ps.Process(i)

        if p.name() == name:
            lista_pid.append(str(i))

    if len(lista_pid) > 0:
        return lista_pid

    else:
        return "não está executando no momento."
```

Tem a finalidade de retornar o PID de um processo com base no nome do mesmo, o método recebe um parâmetro ***name*** que representa nome do processo que o usuário escolheu.

São criadas as variáveis ***lp*** e ***lista_pid*** que representam todos os pid's de processos sendo executados no momento e uma lista vazia que será preenchida com os pid's existentes do processo escolhido pelo usuário. é feita uma verificação da lista no final caso não haja pid nenhum é retornado uma mensagem informando ao cliente que não há execução do processo escolhido.

getProcessInformation()

```python
# Retorna os dados de um PID
def getProcessInformation(pid):
    return ps.Process(int(pid))
```

Tem a finalidade de retornar os dados de um processo com base no PID do mesmo, o método recebe um parâmetro ***pid*** que representa o PID do processo que o usuário escolheu.

sendResponse()

```python
def sendResponse(res):
    # Verifica se a resposta ja é uma lista.
    if type(res) is list:
        # Prepara a lista para o envio
        bytes_resp = pickle.dumps(res)

        # Envia os dados
        cliente.send(bytes_resp)

    else:
        # Gera a lista de resposta
        resposta = []

        # Apenda a resposta para o array de respostas
        resposta.append(res)

        # Prepara a lista para o envio
        bytes_resp = pickle.dumps(resposta)

        # Envia os dados
        cliente.send(bytes_resp)
```

Tem a finalidade de enviar a resposta do servidor para o cliente, foi externalizada em uma função para melhor entendimento do código e menos repetição do mesmo.

Recebe como parâmetro ***res*** que representa a resposta a ser enviada, faz uma verificação do tipo do conteudo da variavel res, caso seja uma lista, sera convertida em bytes atravez do metodo ***pickle.dumbps()*** e enviada para o cliente com o método ***cliente.send()***. Caso não seja um array a resposta será inserida em uma lista, convertida para bytes e enviada para o cliente.

Da estrutura de looping:

Foi escolhido uma estrutura de looping feita com ***while*** para capturar as mensagens

enviadas do cliente, no looping foi feita uma estrutura de decisões com ***if's*** para gerenciar as opções e responder corretamente o cliente.

```
while True:
    # Recebe pedido do cliente:
    msg = cliente.recv(2048)

    print('Connection handled')
```

O looping tem como parâmetro o condicional ***True*** o mantendo enquanto for verdadeiro,

foi criada uma variável ***msg*** que recebe a mensagem do cliente com o método ***cliente.recv()*** passando o máximo de bytes que a mensagem deve ter, no caso 2048, e será informado no console quando a conexão for estabelecida.

Das estruturas de decisão:

As mensagens recebidas do cliente serão decodificadas e respeitarão uma estrutura de decisão como já foi dito.

```
if msg.decode('ascii') == 'fim':
    break
```

A primeira delas, decodifica a mensagem recebida e caso ela seja a ***string "fim"*** ela para o looping de repetição e encerra o lado do servidor.

```
# Percentual de memoria RAM
if msg.decode('ascii') == '1':
    res = []
    res = getMemoryUsagePercent()
    sendResponse(res)
```

A segunda, decodifica a mensagem recebida e caso ela seja a ***string "1",*** terá o objetivo de retornar o percentual de memória RAM usada, será criada uma variável **res** que receberá o retorno da função ***getMemoryUsagePercent()*** e será enviada como resposta para o cliente através do método ***sendResponse()***.

```
# Percentual de uso da CPU
if msg.decode('ascii') == '2':
    res = []
    for i in range(0, ps.cpu_count()):
        temp = getCpuUsagePercent(i)
        res.append(temp)

    sendResponse(res)
```

A terceira, decodifica a mensagem recebida e caso ela seja a ***string "2",*** terá o objetivo de retornar o percentual de cpu usada por núcleo, será criada uma variável ***res***, para contar os núcleos e ter a resposta dividia por eles, é usado uma iteração em cima da quantidade dos mesmos na máquina, será criada uma variável temporária que receberá o retorno da função ***getCpuUsagePercent()*** e será inserida na lista res criada anteriormente que ao sair da iteração será enviada como resposta para o cliente através do método ***sendResponse()***.

```
# Percentual de uso de Disco
if msg.decode('ascii') == '3':
    res = []
    res = ps.disk_usage('.')
    sendResponse(res.percent)
```

A quarta, decodifica a mensagem recebida e caso ela seja a ***string "3",*** terá o objetivo de

retornar o percentual de disco usado, será criada uma variável **res** que receberá o retorno da função ***ps.disk_usage()*** passando ***"."*** como parâmetro e será enviada como resposta para o cliente através do método ***sendResponse()***.

```
# Informações da CPU
if msg.decode('ascii') == '4':
    res = []
    info = cpuinfo.get_cpu_info()
    res = getCpuInfo(info)
    sendResponse(res)
```

A quinta, decodifica a mensagem recebida e caso ela seja a ***string "4",*** terá o objetivo de retornar as informações de cpu, será criada uma variável **res** que receberá o retorno da função ***getCpuInfo()*** passando a variável ***info***, que receberá as informações de cpu em um dicionario retornado do método ***cpuinfo.get_cpu_info()***, como parâmetro e será enviada como resposta para o cliente através do método ***sendResponse()***.

```
# Ip da maquina
if msg.decode('ascii') == '5':
    dic_interfaces = ps.net_if_addrs()
    machine_ip = dic_interfaces['Ethernet'][1].address
    sendResponse(machine_ip)
```

A sexta, decodifica a mensagem recebida e caso ela seja a ***string "5",*** terá o objetivo de retornar o ip da máquina, será criada uma variável, ***dic_interfaces*** que receberá o retorno da função ***ps.net_if_addrs()*** em seguida é criada a variavel ***machine_ip*** que recebe usa ***dic_interfaces*** e extrai do dicionario o endereço IP da rede ***Ethernet*** que será enviada como resposta para o cliente através do método ***sendResponse()***.

```
# retornar as informações do arquivo
if msg.decode('ascii') == '6':
    path = cliente.recv(1024)
    decode_path = path.decode('ascii')
    list_files = os.listdir(decode_path)
    sendResponse(list_files)
```

A sétima, decodifica a mensagem recebida e caso ela seja a ***string "6",*** terá o objetivo de retornar as informações de um arquivo com base em um caminho inserido pelo usuário, após identificar a operação, o servidor ficará em aguardando a entrada do caminho pelo

usuário, recebendo, será decodificado e armazenado na variável ***decode_path***, em seguida será armazenado na variável ***list_files*** a lista do conteúdo existente no caminho que o usuário inseriu retornada pelo método ***os.listdir()*** passando como parâmetro a variável ***decode_path***, por fim será enviada como resposta para o cliente através do método ***sendResponse()***.

```
# retornar as informações de um diretorio
if msg.decode('ascii') == '7':
    path = cliente.recv(1024)
    decode_path = path.decode('ascii')

    response = []

    # Retorna o tamanho do diretorio
    dir_size = getDirSize(decode_path)
    response.append(dir_size)

    # Data de criação do repositorio
    create_date = os.path.getmtime(path)
    response.append(create_date)

    sendResponse(response)
```

A oitava, decodifica a mensagem recebida e caso ela seja a ***string "7",*** terá o objetivo de retornar as informações de um diretório com base em um caminho inserido pelo usuário, após identificar a operação, o servidor ficará em aguardando a entrada do caminho pelo usuário, recebendo, será decodificado e armazenado na variável ***decode_path***, em seguida será criada três variáveis, ***response*** para armazenar a resposta a ser enviada, ***dir_size*** que armazena o retorno do método ***getDirSize()*** passando ***decode_path*** como parâmetro que terá o resultado inserido na lista ***response*** e ***create_date*** que receberá o retorno do método ***os.path.getTime()*** passando o caminho inserido pelo usuário, por final será inserido na lista ***response*** que será enviada como resposta para o cliente através do método ***sendResponse()***.

```
# retornar o PID de um processo
if msg.decode('ascii') == '8':
    process_name = cliente.recv(1024)
    decode_name = process_name.decode('ascii')

    response = []
    # Retorna o pid do processo
    pid = getPIDByName(decode_name)
    response.append(pid[0])

    # Retorna as informações do pid
    process_info = getProcessInformation(int(pid[0]))
    response.append(process_info)

    sendResponse(response)
```

A nona, decodifica a mensagem recebida e caso ela seja a ***string "8",*** terá o objetivo de retornar as informações de um processo com base em um nome de processo inserido pelo usuário, após identificar a operação, o servidor ficará em aguardando a entrada do nome do processo pelo usuário, recebendo, será decodificado e armazenado na variável ***decode_name***, em seguida será criada três variáveis, ***response*** para armazenar a resposta a ser enviada, ***pid*** que armazena o retorno do método ***getPIDByName()*** passando ***decode_name*** como parâmetro que terá o resultado inserido na lista ***response*** e ***process_info*** que receberá o retorno do método ***getProcessInformation()*** passando o PID inserido pelo usuário, por final será inserido na lista ***response*** que será enviada como resposta para o cliente através do método ***sendResponse()***.

```
# Retorna as informações da rede em que a maquina se encontra
if msg.decode('ascii') == '9':
    interfaces = ps.net_if_addrs()
    sendResponse(interfaces)
```

A decima, decodifica a mensagem recebida e caso ela seja a ***string "9",*** terá o objetivo de retornar as informações de redes da máquina em questão, será criada uma variável ***interfaces*** que receberá o retorno da função ***os.net_if_addrs()*** e será enviada como resposta para o cliente através do método ***sendResponse()***.

Por final, Caso o usuário insira “***fim”***, com o break do looping, o socket do servidor será fechado com o método ***socket_servidor.close()***

```
# Fecha socket do servidor
socket_servidor.close()
```

Resultados e Análise

A seguir apresentarei por meio de imagens o funcionamento do sistema, passarei por todas as dez opções contidas no projeto.

Opção um:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> 1

        Porcentagem de memoria usada: 65.4 %
```

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50726)
Connection handled
```

Analise:

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender.

Opção dois:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> 2

        Porcentagem de CPU no nucleo 1 usada: 6.2 %

        Porcentagem de CPU no nucleo 2 usada: 7.6 %

        Porcentagem de CPU no nucleo 3 usada: 6.2 %

        Porcentagem de CPU no nucleo 4 usada: 1.6 %
```

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50753)
Connection handled
```

Analise:

A opção dois atende as espetativas, consegue trazer os dados com precisão, porem acho que poderia ser feita de uma forma onde fosse feita uma busca por chamada e funcionasse encima de uma forma de schedule de alguns segundos ao invez de um time.sleep().

Opção três:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> 3

        Porcentagem de disco usada: 68.7 %
```

Servidor:

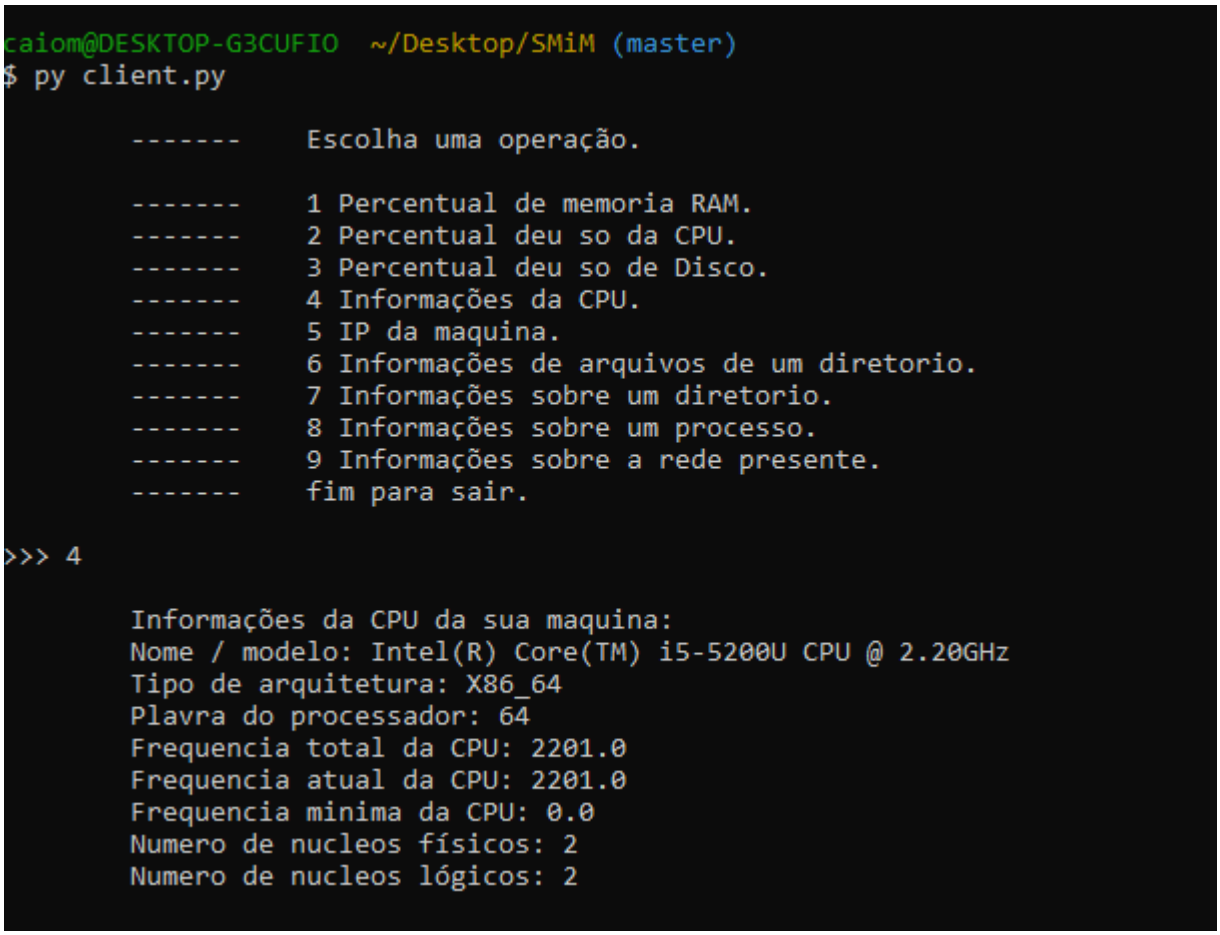
```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50770)
Connection handled
```

Analise

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender.

Opção quatro

Cliente:


caiom@DESKTOP-G3CUFIO ~/Desktop/SMiM (master)
$ py client.py
------- Escolha uma operação.
------- 1 Percentual de memoria RAM.
------- 2 Percentual deu so da CPU.
------- 3 Percentual deu so de Disco.
------- 4 Informações da CPU.
------- 5 IP da maquina.
------- 6 Informações de arquivos de um diretorio.
------- 7 Informações sobre um diretorio.
------- 8 Informações sobre um processo.
------- 9 Informações sobre a rede presente.
------- fim para sair.
>>> 4
Informações da CPU da sua maquina:
Nome / modelo: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
Tipo de arquitetura: X86_64
Plavra do processador: 64
Frequencia total da CPU: 2201.0
Frequencia atual da CPU: 2201.0
Frequencia minima da CPU: 0.0
Numero de nucleos físicos: 2
Numero de nucleos lógicos: 2

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50772)
Connection handled
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "C:\Users\caiom\AppData\Local\Programs\Python\Python36\lib\multiprocessing\spawn
.py", line 105, in spawn_main
    exitcode = _main(fd)
  File "C:\Users\caiom\AppData\Local\Programs\Python\Python36\lib\multiprocessing\spawn
.py", line 114, in _main
    prepare(preparation_data)
  File "C:\Users\caiom\AppData\Local\Programs\Python\Python36\lib\multiprocessing\spawn
.py", line 225, in prepare
    _fixup_main_from_path(data['init_main_from_path'])
  File "C:\Users\caiom\AppData\Local\Programs\Python\Python36\lib\multiprocessing\spawn
.py", line 277, in _fixup_main_from_path
    run_name="__mp_main__")
  File "C:\Users\caiom\AppData\Local\Programs\Python\Python36\lib\runpy.py", line 263,
in run_path
    pkg_name=pkg_name, script_name=fname)
  File "C:\Users\caiom\AppData\Local\Programs\Python\Python36\lib\runpy.py", line 96, i
n _run_module_code
    mod_name, mod_spec, pkg_name, script_name)
  File "C:\Users\caiom\AppData\Local\Programs\Python\Python36\lib\runpy.py", line 85, i
n _run_code
    exec(code, run_globals)
  File "C:\Users\caiom\Desktop\SMiM\server.py", line 23, in <module>
    socket_servidor.bind((host, porta))
OSError: [WinError 10048] Normalmente é permitida apenas uma utilização de cada endereç
o de soquete (protocolo/endereço de rede/porta)
```

Analise:

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender, porém como visto na imagem do servidor, há um erro que eu nao consegui resolver, ele não interfere no processamento do dado ou para a aplicação de algum jeito, uma issue foi aberta no github com o erro e será resolvida da melhor forma possível.

Opção cinco:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> 5

        Ip da maquina: 169.254.54.200
```

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50827)
Connection handled
```

Analise:

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender.

Um ponto de melhoria, seria retornar todos os endereços das redes em que a máquina se encontra.

Opção seis:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> 6

        Por favor, digite o caminho do arquivo.

>>> c:

        Tamanho    Data de Modificação        Data de Criação            Nome
        0.01 KB    Fri Sep 21 23:19:45 2018   Fri Sep 21 23:19:39 2018   .gitignore
        9.30 KB    Sat Sep 22 17:06:22 2018   Fri Sep 21 23:16:38 2018   client.py
        0.04 KB    Tue Sep 18 08:09:07 2018   Tue Sep 18 08:09:07 2018   README.md
        5.16 KB    Sun Sep 23 16:13:38 2018   Fri Sep 21 23:16:38 2018   server.py
```

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50833)
Connection handled
```

Analise

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender.

Um ponto de melhoria, seria uma melhor captação da mensagem do usuário junto com um melhor tratamento de erro, talvez um dicionário com os possíveis erros mapeados e identificados por alguma função na hora de lançar a ***exception***.

Opção sete:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> 7

        Por favor, digite o caminho de um diretorio.

>>> c:\Users

        Tamanho          Data de Criação              Nome do diretorio
        0.17 KB          Sun Sep 16 16:31:43 2018     Users
```

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50857)
Connection handled
```

Analise:

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender.

Um ponto de melhoria seria trazer mais informações do diretório em questão porém a disposição das informações para o usuário deve ser pensada melhor para proporcionar uma melhor experiência.

Opção oito:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> 8

        Por favor, digite o nome do processo.

>>> Telegram.exe

        PID 12324
        Nome: Telegram.exe
        Executável: C:\Users\caiom\AppData\Roaming\Telegram Desktop\Telegram.exe
        Tempo de criação: Sun Sep 23 13:30:15 2018
        Tempo de usuário: 87.375 s
        Tempo de sistema: 38.140625 s
        Percentual de uso de CPU: 0.0 %
        Percentual de uso de memória: 0.55 %
        Uso de memória: 44.89 MB
        Número de threads: 13
```

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50862)
Connection handled
```

Analise:

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender.

Um ponto de melhoria seria dar a opção de mostrar os processos existentes para o usuario, pos nao necessariamente o usuario tem conhecimento dos nomes do processos e um outro ponto seria um melhor tratamento de erro como falado anteriormente um dicionário prevendo os possíveis erros e ao acontecer identificaria nesse dicionário o erro com base em algum identificador gerado pelo erro.

Opção nove:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> 9

Ethernet:
        Endereço IP: F8-A9-63-61-10-11
        Familia: AddressFamily.AF_LINK
        Netmask: None

        Endereço IP: 169.254.54.200
        Familia: AddressFamily.AF_INET
        Netmask: 255.255.0.0

        Endereço IP: fe80::c989:af19:b545:36c8
        Familia: AddressFamily.AF_INET6
        Netmask: None
```

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50865)
Connection handled
```

Analise:

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender.

Um ponto de melhoria seria uma melhor disposição das informações para o usuário, em minha opinião não ficou disposto de uma forma agradável.

Opção dez:

Cliente:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py client.py

        -------    Escolha uma operação.

        -------    1 Percentual de memoria RAM.
        -------    2 Percentual deu so da CPU.
        -------    3 Percentual deu so de Disco.
        -------    4 Informações da CPU.
        -------    5 IP da maquina.
        -------    6 Informações de arquivos de um diretorio.
        -------    7 Informações sobre um diretorio.
        -------    8 Informações sobre um processo.
        -------    9 Informações sobre a rede presente.
        -------    fim para sair.

>>> fim
Pressione qualquer tecla para sair...
```

Servidor:

```
caiom@DESKTOP-G3CUFIO  ~/Desktop/SMiM (master)
$ py server.py
Servidor de nome DESKTOP-G3CUFIO esperando conexão na porta 14562
Conectado a: ('192.168.0.106', 50903)
Connection handled
```

Analise:

A opção um atinge as expectativas, tem uma ótima performance, e o código é bem simples de entender.

Conclusão

A experiência de construir um sistema como esse, para mim foi algo no mínimo interessante, podemos ver a linguagem escolhida sendo usada para capturar informações de uma máquina de forma fácil e simples, as bibliotecas que auxiliam sao bem faceis de manusear e simples de entender.

A dificuldade da construção do sistema foi aprender a lidar com as informações de máquina necessárias, porém com todo o andamento dos estudos durante o período, se tornou um desafio agradável.

Com tudo o que foi visto concluo que pude aprender a gerenciar um projeto de pequena escala, pensar um pouco mais de forma analita aos problemas que a mim foram apresentados e absorvi certos conhecimentos do assunto proposto. O projeto tem algumas deficiências como tratamento de erro e disposição das informações para o usuário, porém imagino que sejam fáceis de solucionar, acredito que vale sim apena investir mais tempo e estudo no projeto para melhorá-lo, o projeto atendeu sim as minhas expectativas e acredito que deva dar continuidade.