

typewriter

Documentação

Projeto de Bloco - Arquitetura de Computadores, Sistemas Operacionais e Redes

Instituto Infnet - ESTI - Engenharia da Computação

Camila Costa, Christian Rocha, Christian Tavares, Thiago Rios

Prof. Cassius Figueiredo

28/03/2019

Sumário:

Introdução.....	3
Desenvolvimento.....	4
Recursos.....	11
Conclusão.....	17
Código Cliente.....	18
Código Servidor.....	22
Referências.....	29
Glossário.....	29

Introdução:

Relatório contendo o desenvolvimento da aplicação subdividido por etapas que foram realizadas nos últimos meses baseado em testes e implementações da aplicação desenvolvida em Python através do compilador Thonny.

Aplicação de monitoramento de recursos – processador, memória, armazenamento, arquivos, processos e rede – de um sistema computacional remoto baseado em arquitetura Cliente-Servidor, onde o Cliente requisita informações sobre o atual uso de determinado recurso do sistema computacional e o Servidor retorna essa informação para o Cliente e apresenta-os por meio da Command Prompt (CMD) ou do Power Shell do sistema operacional Windows.

Desenvolvimento

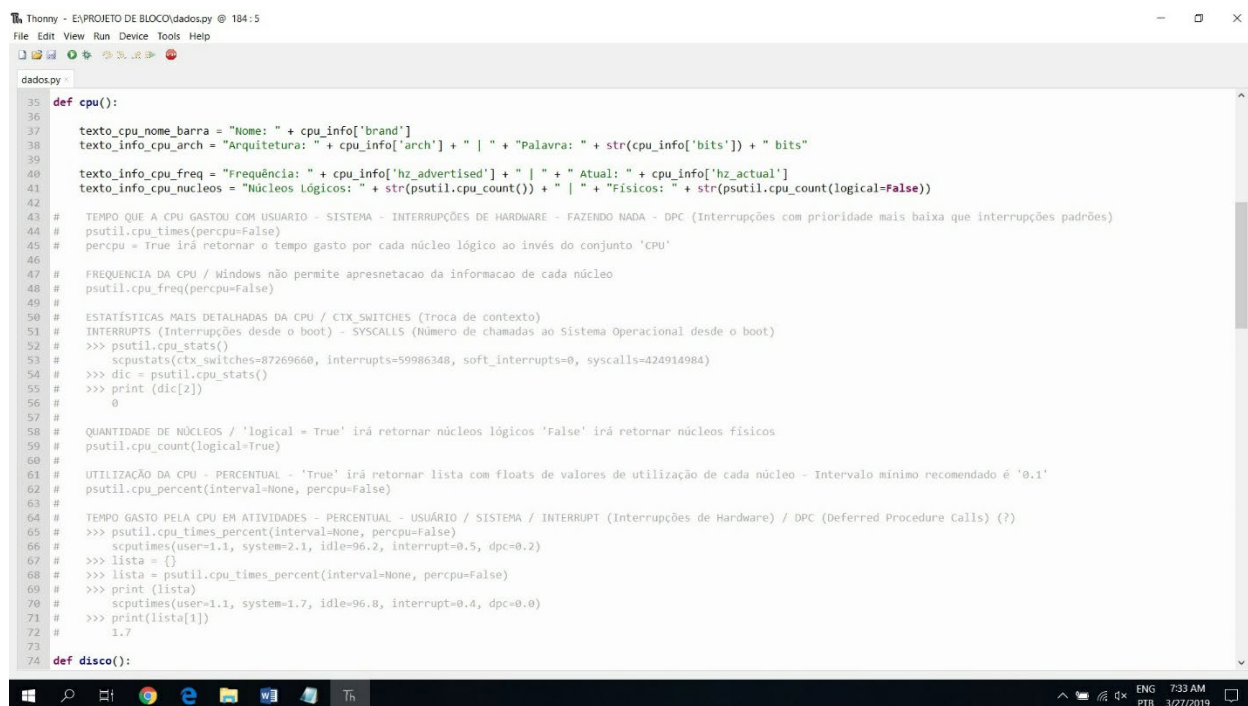
Etapas 1 – Seleção das funções:

Para obter as informações da máquina servidora foram utilizadas as seguintes bibliotecas externas no compilador Thonny junto com as bibliotecas nativas do Python para maior suporte:

- psutil (v 5.5.0) ^[1]
- Py-cpuinfo (v 3.2.0) ^[2]
- nmap (v 0.6.1) * ^[3]
- datetime, os, pickle, platform, socket, subprocess e time (nativos do Python)

**: Para suporte à biblioteca nmap foi utilizado o software nmap (v 7.7.0) ^[4] no sistema operacional Windows 10 (64 bits).*

Devido ao fato de que cada biblioteca externa é utilizada no desenvolvimento de aplicações em diversos sistemas operacionais e cada sistema fornecer seu próprio nível de acesso, características distintas e quantidade variável de informações disponíveis para coleta, houve a necessidade de um levantamento completo de cada ferramenta disponível em cada biblioteca externa para o sistema que foi feito o desenvolvimento da aplicação, no caso Windows, sistema operacional de código fonte fechado. Por se tratar de um sistema com acesso mais restrito que o Linux / UNIX, a quantidade de informação disponível foi mais limitada, porém não apareceu um agravante visto que as informações básicas de monitoramento de recursos estavam disponíveis.



```
dados.py
35 def cpu():
36
37     texto_cpu_nome_barra = "Nome: " + cpu_info['brand']
38     texto_info_cpu_arch = "Arquitetura: " + cpu_info['arch'] + " | " + "Palavra: " + str(cpu_info['bits']) + " bits"
39
40     texto_info_cpu_freq = "Frequência: " + cpu_info['hz_advertised'] + " | " + "Atual: " + cpu_info['hz_actual']
41     texto_info_cpu_nucleos = "Núcleos Lógicos: " + str(psutil.cpu_count()) + " | " + "Físicos: " + str(psutil.cpu_count(logical=False))
42
43 # TEMPO QUE A CPU GASTOU COM USUARIO - SISTEMA - INTERRUPÇÕES DE HARDWARE - FAZENDO NADA - DPC (Interrupções com prioridade mais baixa que interrupções padrões)
44 # psutil.cpu_times(percpu=False)
45 # percpu = True irá retornar o tempo gasto por cada núcleo lógico ao invés do conjunto 'cpu'
46
47 # FREQUENCIA DA CPU / Windows não permite apresnetacao da informacao de cada núcleo
48 # psutil.cpu_freq(percpu=False)
49 #
50 # ESTATÍSTICAS MAIS DETALHADAS DA CPU / CTX_SWITCHES (Troca de contexto)
51 # INTERRUPTS (Interrupções desde o boot) - SYSCALLS (Número de chamadas ao Sistema Operacional desde o boot)
52 # >>> psutil.cpu_stats()
53 # >>> scputimes(ctx_switches=87269660, interrupts=59986348, soft_interrupts=0, syscalls=424914984)
54 # >>> dic = psutil.cpu_stats()
55 # >>> print(dic[2])
56 #
57 #
58 # QUANTIDADE DE NÚCLEOS / 'logical = True' irá retornar núcleos lógicos 'false' irá retornar núcleos físicos
59 # psutil.cpu_count(logical=True)
60 #
61 # UTILIZAÇÃO DA CPU - PERCENTUAL - 'True' irá retornar lista com floats de valores de utilização de cada núcleo - Intervalo mínimo recomendado é '0.1'
62 # psutil.cpu_percent(interval=None, percpu=False)
63 #
64 # TEMPO GASTO PELA CPU EM ATIVIDADES - PERCENTUAL - USUÁRIO / SISTEMA / INTERRUPT (Interrupções de Hardware) / DPC (Deferred Procedure Calls) (?)
65 # >>> psutil.cpu_times_percent(interval=None, percpu=False)
66 # >>> sputimes(user=1.1, system=2.1, idle=96.2, interrupt=0.5, dpc=0.2)
67 # >>> lista = {}
68 # >>> lista = psutil.cpu_times_percent(interval=None, percpu=False)
69 # >>> print(lista)
70 # >>> sputimes(user=1.1, system=1.7, idle=96.8, interrupt=0.4, dpc=0.0)
71 # >>> print(lista[1])
72 #
73 #
74 def disco():
```

Figura 1: Levantamento de informações para função do Processador.

```
Thonny - E:\PROJETO DE BLOCO\dados.py @ 104:5
File Edit View Run Device Tools Help

dados.py
74 def disco():
75
76 # RETORNA TODOS OS DISCOS FIXOS E REMOVÍVEIS - Não funciona o 'True' no Windows então tanto faz o parâmetro dentro do parêntesis
77 # psutil.disk_partitions(all=False)
78 #
79 # [sdiskpart(device='C:\\', mountpoint='C:\\', fstype='NTFS', opts='rw,fixed'),
80 #  sdiskpart(device='D:\\', mountpoint='D:\\', fstype='exFAT', opts='rw,fixed'),          <----- Posição 1 - buscar por chave 'fstype' ou desejada
81 #  sdiskpart(device='E:\\', mountpoint='E:\\', fstype='NTFS', opts='rw,fixed'),
82 #  sdiskpart(device='H:\\', mountpoint='H:\\', fstype='NTFS', opts='rw,removable')]
83 #
84 # dicionario = []
85 # dicionario = psutil.disk_partitions(all=False)
86 # print(dicionario[1].fstype)
87 # 'exFAT'          <----- Posição 1 - chave 'fstype' impressa
88 #
89 # VALOR DE UTILIZAÇÃO DOS DISCOS
90 # psutil.disk_usage('/') <- Tem que ter '/' senão irá dar erro - Retorna valor em
91 #
92 # dicionario = psutil.disk_usage('C:/')
93 # print(dic)
94 # sdiskusage(total=203896647680, used=126147117056, free=77749530624, percent=61.9)
95 #
96 # >>> dic = psutil.disk_usage('D:/')
97 # >>> print(dic)
98 # sdiskusage(total=1000169537536, used=968121384960, free=32048152576, percent=96.8)
99 # esp = dic[0]
100 # >>> print(esp)
101 # 1000169537536
102 # >>> espaco = esp / (1024 * 1024 * 1024)
103 # >>> print(espaco)
104 # 931.48046875
105 # >>> espaco = round(esp / (1024 * 1024 * 1024), 2)
106 # >>> print(espaco)
107 # 931.48          <--- Valor final de espaço interno total do HD já em 'GB'
108 #
109 # ESTATÍSTICAS DE I/O DO HD - READ_COUNT (número de leituras) - WRITE_COUNT (número de gravações)
110 # READY_BYTES (número de bytes lidos) - WRITE_BYTES(número de bytes escritos)
111 # READ_TIME (tempo gasto em 'ms' lendo o disco) - WRITE_TIME (tempo gasto em 'ms' escrevendo no disco)
112 #
113 # psutil.disk_io_counters(perdisk=False, nowrap=True)
```

Figura 2: Levantamento de informações para função do Disco Rígido.

Etapa 2 – Teste das funções:

Máquina de teste 1:

- Processador: Intel Core i7-7500U (Base: 2.90 GHz / Testes: 3.5 GHz)
- Memória RAM: 8 GB / 2133 MHz / SODIMM
- Armazenamento: SanDisk Ultra II (SSD)
- Rede Ethernet: Realtek PCIe GBE Family Controller
- Rede Wireless: Qualcomm Atheros QCA9377

Máquina de teste 2:

- Processador: Intel Core i7-7700 (Base: 3.60 GHz)
- Memória RAM: 8 GB
- Armazenamento: SSD 250 GB

Máquina de teste 3:

- Processador: AMD A6-3500
- Memória RAM: 8 GB
- Armazenamento: HDD com 1 TB
- Rede Wireless: D-Link DWA-131 Wireless Nano Adapter

O projeto começou a tomar forma com a seleção das funções feita previamente, o que demorou tempo, porém resultou em um foco maior em quais funções realmente deveriam ser abordadas na aplicação para que fosse funcional e realmente útil o resultado apresentado para o usuário.

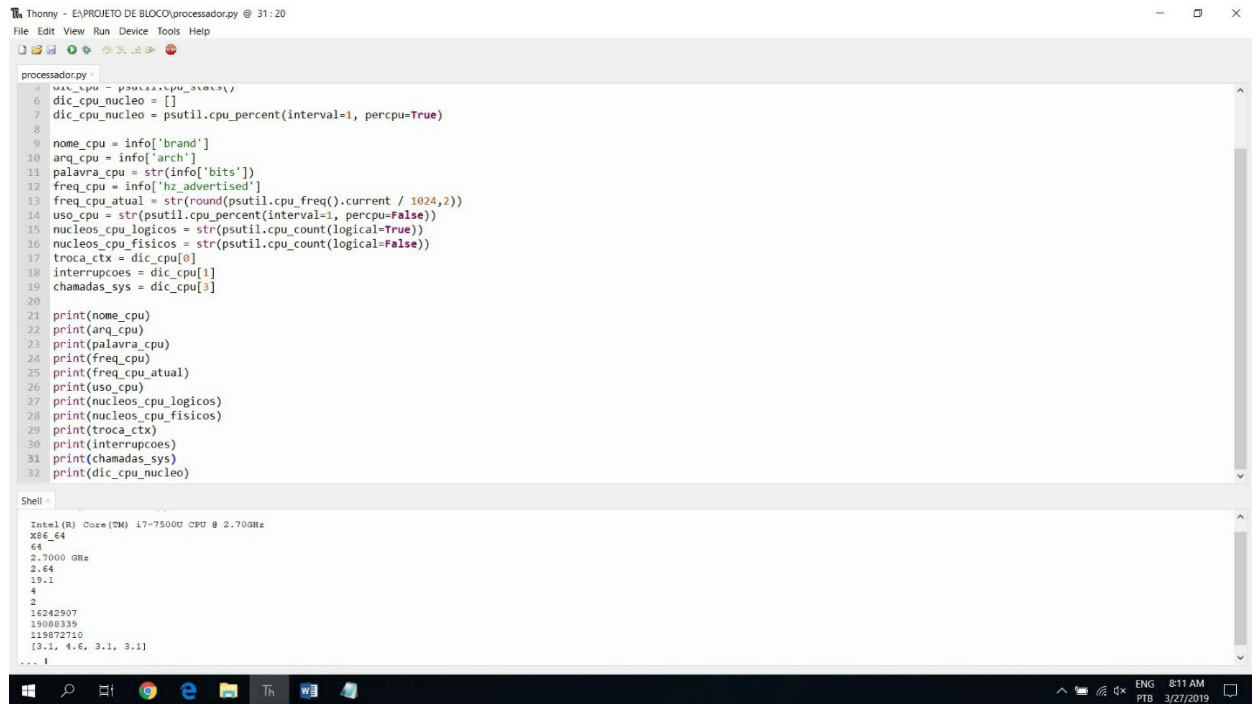
Foi decidido que iniciariam os testes apenas com resultado no Cliente, buscando informações do próprio computador para facilitarem os testes e resolverem possíveis erros. Sem dúvida não foram utilizadas todas as funções levantadas na Etapa 1 visto que seu número é consideravelmente grande e por possuir funções que não são relevantes para um servidor – E.g.: função para medição de temperatura da bateria (notebook) – o que resultou em uma gama de funções ainda mais estrita.

Houveram funções – E.g.: tempo gasto pelo processador em interrupções e DPC, estatística de I/O do disco rígido, quantidade de bytes escritos e lidos, tipo de conexão do NIC – que poderiam ter sido implementadas, porém foram cortadas para manter a essência da aplicação que seria o simples monitoramento dos recursos de um sistema computacional remoto. Da mesma forma, funções que não funcionaram como esperado ou houveram divergência de execução nas máquinas que foram feitos os testes foram retiradas também.

Os testes se resumiram à coleta de uma determinada informação e a impressão dela no console do compilador. Os recursos monitorados foram separados em projetos específicos para testes unitários de cada recurso e suas respectivas funções, após foram testados como

um todo e assim reuniram-se todos as áreas monitoradas para verificar a execução em conjunto e continuar a implementação.

Houveram resultados que são retornados por meio de listas e dicionários e para isso sua manipulação foi feita para extração de uma ou mais informação(ões) que entraria(m) no projeto final.



```
Thonny - E:\PROJETO DE BLOCO\processador.py @ 31:20
File Edit View Run Device Tools Help

processador.py
1 max_cpu = psutil.cpu_count()
2 dic_cpu_nucleo = {}
3 dic_cpu_nucleo = psutil.cpu_percent(interval=1, percpu=True)
4
5
6 nome_cpu = info['brand']
7 arq_cpu = info['arch']
8 palavra_cpu = str(info['bits'])
9 freq_cpu = info['hz_advertised']
10 freq_cpu_atual = str(round(psutil.cpu_freq().current / 1024,2))
11 uso_cpu = str(psutil.cpu_percent(interval=1, percpu=False))
12 nucleos_cpu_logicos = str(psutil.cpu_count(logical=True))
13 nucleos_cpu_fisicos = str(psutil.cpu_count(logical=False))
14 troca_ctx = dic_cpu[0]
15 interrupcoes = dic_cpu[1]
16 chamadas_sys = dic_cpu[3]
17
18 print(nome_cpu)
19 print(arq_cpu)
20 print(palavra_cpu)
21 print(freq_cpu)
22 print(freq_cpu_atual)
23 print(uso_cpu)
24 print(nucleos_cpu_logicos)
25 print(nucleos_cpu_fisicos)
26 print(troca_ctx)
27 print(interrupcoes)
28 print(chamadas_sys)
29 print(dic_cpu_nucleo)
30
31
32

Shell
Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
X86_64
64
2.7000 GHz
2.64
19.1
4
2
16242907
19088339
119872710
[3.1, 4.6, 3.1, 3.1]
...
I
```

Figura 3: Teste do recurso Processador.

Etapas 3 – Implementação:

Foram separados os projetos, o Servidor é agora responsável apenas pela captura das informações dos seus recursos e envio do conjunto de dados (brutos) através de lista ou dicionário (dependendo do recurso escolhido) para o Cliente com a conversão desses objetos retornados pela função do recurso em bytes, permitido com a utilização da biblioteca nativa do Python chamada “pickle” que realiza essa conversão para assim ser possível enviar esse objeto agora em bytes pela conexão criada pela biblioteca nativa do Python definida por “socket” que realiza esta comunicação entre os dois lados sob o protocolo escolhido que foi o TCP. O Cliente é responsável pelo recebimento do pacote de resposta ainda em bytes e a mesma biblioteca “pickle” citada previamente realiza a conversão de bytes para o objeto gerado pela função no servidor. O cliente processa a lista ou dicionário recebido, formata a resposta e imprime-a no meio de apresentação, podendo ser compilador, CMD ou Power Shell no Windows.

Nesta Etapa ocorreu a criação do código para conexão entre o Cliente e Servidor. Foram feitos para ambos protocolos TCP e UDP, porém escolhido o TCP por sua confiabilidade na entrega da informação. Houveram as avaliações de performance embora o UDP seja mais rápido. Para pacotes de até 32 kilobyte (KB) foi tomado por base indiferente a performance dos protocolos, o que resultou na escolha de um protocolo mais confiável com o TCP. Não houve problema na criação do socket.

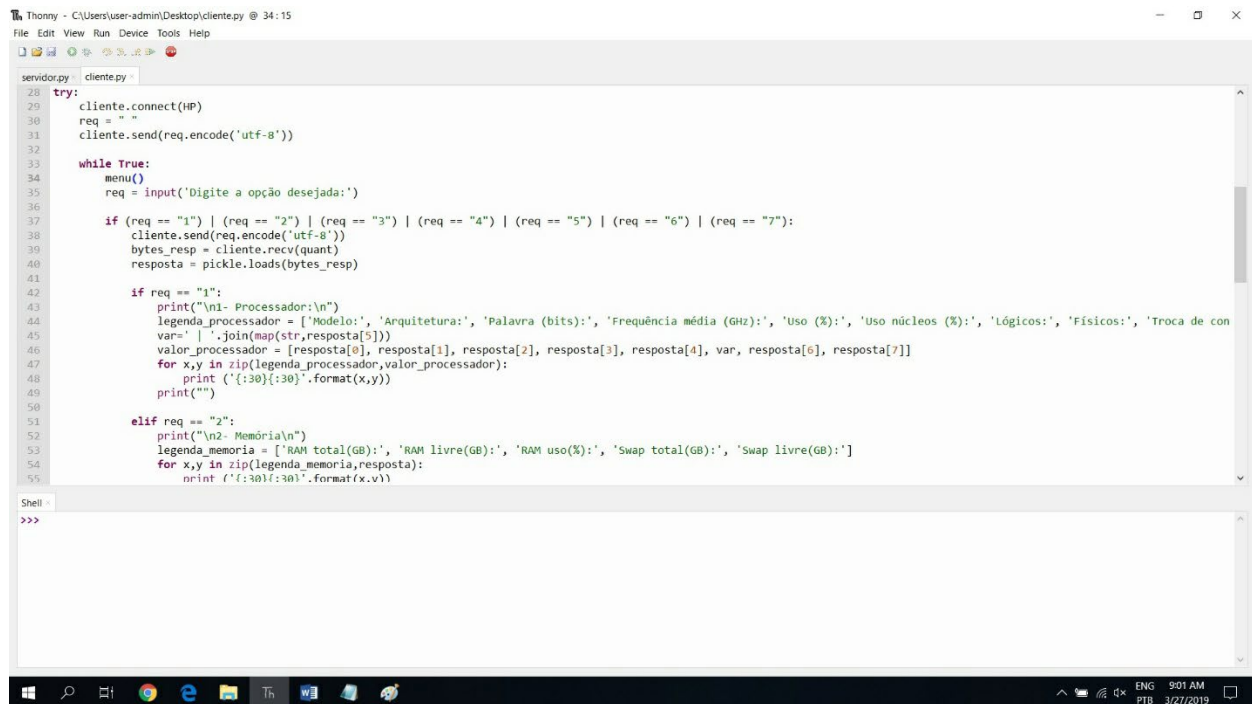
A maior dificuldade na parte de formatação foi extrair da resposta os resultados, visto que algumas respostas vinham como lista, porém com outra lista e um dicionário dentro.

Funções como limpeza da tela através do comando “cls” na CMD tiveram que ser implementadas junto à utilização da biblioteca nativa do Python “time” com a função “time.sleep(*tempo em segundos*)” para aparecer na tela o resultado e aguardar alguns segundos para limpar a tela e mostrar o menu novamente, pois aparecia a informação e a tela era limpa em seguida, para evitar maiores problemas com implementação de alguma tecla, foi escolhido manter a função “time” para resolver o problema e o tempo definido para o programa aguardar para limpar a tela foi de 15 (quinze) segundos.

A opção do recurso de processos (5) apresentou problema ao final da implementação, onde a conexão é encerrada por parte do cliente após execução da requisição enviada pelo cliente e respondida com os dados dos processos pelo servidor. Da mesma forma, ocorre a impressão da variável com o valor de todos os processos coletados ao final da execução da requisição.

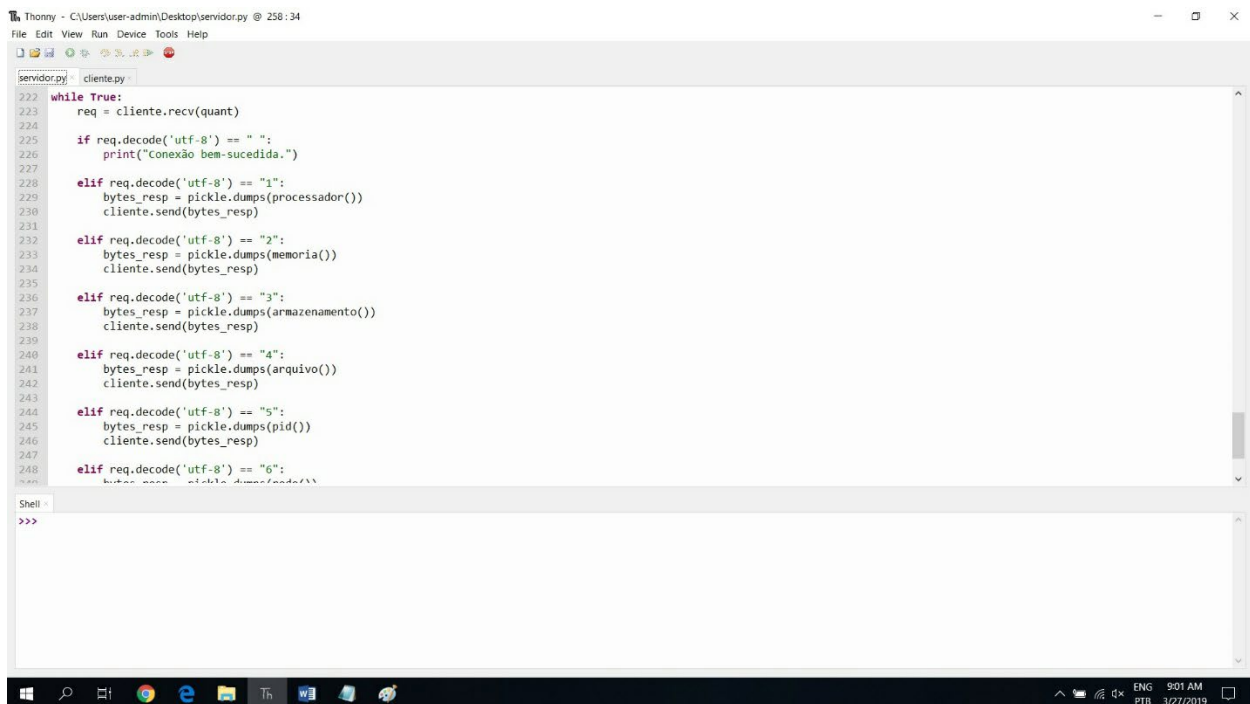
A maior dificuldade do projeto foi sem dúvida a parte do nmap, onde diversos erros surgiram e precisamos aprender a interpretá-los e resolvê-los. A função consome uma quantidade consideravelmente alta de tempo para retornar o resultado pois o teste é executado por completo, verificando as sub-redes e portas abertas dentro de cada uma delas. Ao final, observa-se que o mesmo erro com a função de processos se repete, porém não conseguimos identificar o motivo.

Outro ponto que não obtivemos êxito foi conseguir implementar um meio de encerrar a aplicação por acionamento de alguma tecla. O único jeito de encerrar a aplicação é por algum erro ou a saída inesperada após a execução da requisição de número cinco (5).



```
servidor.py cliente.py
28 try:
29     cliente.connect(HIP)
30     req = " "
31     cliente.send(req.encode('utf-8'))
32
33 while True:
34     menu()
35     req = input('Digite a opção desejada:')
36
37 if (req == "1") | (req == "2") | (req == "3") | (req == "4") | (req == "5") | (req == "6") | (req == "7"):
38     cliente.send(req.encode('utf-8'))
39     bytes_resp = cliente.recv(quant)
40     resposta = pickle.loads(bytes_resp)
41
42     if req == "1":
43         print("\n1- Processador:\n")
44         legenda_processador = ['Modelo:', 'Arquitetura:', 'Palavra (bits):', 'Frequência média (GHz):', 'Uso (%):', 'Uso núcleos (%):', 'Lógicos:', 'Físicos:', 'Troca de con
45         var= ' | '.join(map(str,resposta[5]))
46         valor_processador = [resposta[0], resposta[1], resposta[2], resposta[3], resposta[4], var, resposta[6], resposta[7]]
47         for x,y in zip(legenda_processador,valor_processador):
48             print('{:30}{:30}'.format(x,y))
49         print("")
50
51     elif req == "2":
52         print("\n2- Memória\n")
53         legenda_memoria = ['RAM total(GB):', 'RAM livre(GB):', 'RAM uso(%):', 'Swap total(GB):', 'Swap livre(GB):']
54         for x,y in zip(legenda_memoria,resposta):
55             print('{:30}{:30}'.format(x,y))
```

Figura 4 (Cliente): Tentativa de conexão com servidor, envio da requisição, recebimento da resposta, decodificação do pacote de resposta, formatação e impressão da resposta.



The screenshot shows a Thonny IDE window with the title bar "Thonny - C:\Users\user-admin\Desktop\servidor.py @ 258 : 34". The menu bar includes "File", "Edit", "View", "Run", "Device", "Tools", and "Help". The main editor displays the code for "servidor.py" with line numbers 222 to 248. The code is a while loop that receives a request from a client, decodes it, and then based on the received number (1-6), calls a function (processador, memoria, armazenamento, arquivo, pid) and sends the result back to the client as bytes. The shell window at the bottom shows a prompt ">>>".

```
222 while True:
223     req = cliente.recv(quant)
224
225     if req.decode('utf-8') == " ":
226         print("Conexão bem-sucedida.")
227
228     elif req.decode('utf-8') == "1":
229         bytes_resp = pickle.dumps(processador())
230         cliente.send(bytes_resp)
231
232     elif req.decode('utf-8') == "2":
233         bytes_resp = pickle.dumps(memoria())
234         cliente.send(bytes_resp)
235
236     elif req.decode('utf-8') == "3":
237         bytes_resp = pickle.dumps(armazenamento())
238         cliente.send(bytes_resp)
239
240     elif req.decode('utf-8') == "4":
241         bytes_resp = pickle.dumps(arquivo())
242         cliente.send(bytes_resp)
243
244     elif req.decode('utf-8') == "5":
245         bytes_resp = pickle.dumps(pid())
246         cliente.send(bytes_resp)
247
248     elif req.decode('utf-8') == "6":
249         bytes_resp = pickle.dumps(quant)
```

Shell >>>

Figura 5 (Servidor): Recebimento da requisição, análise do número escolhido, chamada da função que retornará o uso do recurso desejado, conversão para bytes da resposta e envio para o Cliente.

Recursos

Detalhamento e interpretação dos resultados gerados por cada função do projeto. Cada item corresponde a uma requisição de determinado recurso do servidor. Cada subitem corresponde ao que essa requisição irá retornar para o cliente.

1. Processador:

```
C:\Users\Thiago\Downloads>py cliente_novo.py
=====
* 1- Processador          *
* 2- Memória              *
* 3- Armazenamento        *
* 4- Arquivos e diretórios *
* 5- Processos : PID      *
* 6- Rede                 *
* 7- Sair                  *
=====
Digite a opção desejada:1
1- Processador:
Modelo:          AMD A6-3500 APU with Radeon(tm) HD Graphics
Arquitetura:     X86_64
Palavra (bits):  64
Frequência média (GHz): 1.6
Uso (%):         23.8
Uso núcleos (%): 26.2 : 28.1 : 37.7
Lógicos:        3
Físicos:        3
```

- 1.1. Modelo: Retorna a fabricante, modelo, frequência base anunciada.
- 1.2. Arquitetura: Retorna o tipo de arquitetura do processador.
- 1.3. Palavra: Retorna à quantidade de palavras que é processado em conjunto em um sistema computacional. E.g.: Um processador com palavra no valor de 64 bits consegue processar 8 palavras por vez, visto que uma palavra possui 8 bits ($64 / 8 = 8$).
- 1.4. Frequência média: No momento da execução da função do processador ocorrem dez coletas da frequência atual do processador com intervalos de um segundo entre elas, depois uma média é feita e gerado o valor da frequência média em Giga-hertz (GHz).
- 1.5. Uso: No momento da execução da função do processador ocorrem dez coletas da porcentagem de uso do processador com intervalos de um segundo entre elas, depois uma média é feita e gerado o valor do uso médio em por cento (%).
- 1.6. Uso núcleos: Retorna o uso em por cento de cada núcleo do processador.
- 1.7. Núcleos Físicos: Retorna quantidade de núcleos físicos do processador.
- 1.8. Núcleos Lógicos: Retorna quantidade de núcleos lógicos do processador.

2. Memória:

```
=====
1- Processador
2- Memória
3- Armazenamento
4- Arquivos e diretórios
5- Processos | PID
6- Rede
7- Sair
=====
Digite a opção desejada:2

2- Memória

RAM total(GB):           8
RAM livre(GB):           4
RAM uso(%):              55
Swap total(GB):          15
Swap livre(GB):          7
```

- 2.1. RAM total: Retorna à quantidade de memória total do sistema computacional em gigabyte (GB).
- 2.2. RAM disponível: Retorna à quantidade de memória disponível do sistema computacional em gigabyte (GB).
- 2.3. RAM uso: Retorna à quantidade de memória em uso do sistema computacional em gigabyte (GB).
- 2.4. Swap total: Retorna à quantidade de memória swap total do sistema computacional em gigabyte (GB).
- 2.5. Swap disponível: Retorna à quantidade de memória swap disponível do sistema computacional em gigabyte (GB).

3. Armazenamento:

```
=====
1- Processador
2- Memória
3- Armazenamento
4- Arquivos e diretórios
5- Processos | PID
6- Rede
7- Sair
=====
Digite a opção desejada:3

3- Armazenamento

Total(GB):                223
Livre(GB):                 87
Utilizado(%):              61
```

- 3.1. Total: Retorna à quantidade de armazenamento total na partição corrente do sistema computacional em gigabyte (GB).
- 3.2. Livre: Retorna à quantidade de armazenamento disponível na partição corrente do sistema computacional em gigabyte (GB).
- 3.3. Utilizado: Retorna à quantidade de armazenamento utilizado na partição do sistema operacional em forma de percentual.

4. Arquivos

```
4- Arquivos e Diretórios

Diretórios

My Music
My Pictures
My Videos

Arquivos

cliente.py
Tamanho (bytes): 5
Caminho absoluto: C:\Users\user-admin\Documents\cliente.py
Criação: Fri Mar 29 08:10:03 2019
Última modificação: Fri Mar 29 08:53:54 2019
Último acesso: Fri Mar 29 08:53:54 2019

desktop.ini
Tamanho (bytes): 0
Caminho absoluto: C:\Users\user-admin\Documents\desktop.ini
Criação: Sun Mar 24 14:57:41 2019
Última modificação: Mon Mar 25 13:07:17 2019
Último acesso: Mon Mar 25 13:07:17 2019

servidor.py
Tamanho (bytes): 9
Caminho absoluto: C:\Users\user-admin\Documents\servidor.py
Criação: Fri Mar 29 08:10:03 2019
Última modificação: Fri Mar 29 09:12:10 2019
Último acesso: Fri Mar 29 09:12:10 2019
```

- 4.1. Tamanho: Retorna o tamanho do arquivo em bytes no diretório corrente.
- 4.2. Caminho: Retorna o caminho que o arquivo está localizado.
- 4.3. Criação: Retorna a data de criação do arquivo.
- 4.4. Modificação: Retorna a última data de modificação do arquivo.
- 4.5. Último acesso: Retorna a data do último acesso ao arquivo.

5. Processos

```
PID: 1676
Executável: C:\Users\Thiago\.thonny\BundledPython36\Scripts\python.exe
CPU (s): 0.17
Memória (MB): 13.88
Nome: dwm.exe
PID: 1748
Executável: C:\Windows\System32\dwm.exe
CPU (s): 312.8
Memória (MB): 35.0
Nome: msseces.exe
PID: 1776
Executável: C:\Program Files\Microsoft Security Client\msseces.exe
CPU (s): 0.36
Memória (MB): 14.2
Nome: RAUCpl64.exe
PID: 1800
Executável: C:\Program Files\Realtek\Audio\HDA\RAUCpl64.exe
CPU (s): 0.03
Memória (MB): 10.71
Nome: explorer.exe
PID: 1816
Executável: C:\Windows\explorer.exe
CPU (s): 68.56
Memória (MB): 77.64
Nome: spoolsv.exe
```

- 5.1. PID: Retorna o número identificador do processo no sistema operacional.
- 5.2. Executável: Retorna o caminho absoluto do executável do processo.
- 5.3. CPU: Retorna em segundos a quantidade de tempo que o processador gastou realizando o processamento desse processo criado pelo usuário.
- 5.4. Memória: Retorna à quantidade em megabyte (MB) de memória que o processo utilizou.

6. Rede

```
C:\Users\AlphaBear\Desktop\ProjetoFinal>py cliente.py
=====
1- Processador
2- Memória
3- Armazenamento
4- Arquivos e diretórios
5- Processos | PID
6- Rede
7- Sair
=====
Digite a opção desejada:6

6- Rede

IPv4:          192.168.1.16
IPv6:          fe80::98b8:92ef:477a:1634
Máscara:       255.255.255.0
MAC:           E0-D5-5E-F1-B4-38
```

- 6.1. IPv4: Retorna o IP do servidor no formato IPv4.
- 6.2. IPv6: Retorna o IP do servidor no formato IPv6.
- 6.3. Máscara: Retorna a máscara que o servidor está utilizando.
- 6.4. MAC: Retorna o endereço MAC do servidor.

7. Nmap

```
C:\Users\user-admin\Documents>py cliente.py
=====
1- Processador
2- Memória
3- Armazenamento
4- Arquivos e diretórios
5- Processos | PID
6- Rede
7- Nmap
=====
Digite a opção desejada:7
Entre com o número do IP da rede: 186.192.90.

7- NMAP

186.192.90.3
Protocolo: TCP
Porta: 443
Estado: open

186.192.90.5
Protocolo: TCP
Porta: 443
Estado: open
```

7.1. IP (sub-rede): Retorna o IP da sub-rede.

7.2. Protocolo: Retorna o protocolo da conexão (E.g.: TCP ou UDP).

7.3. Porta: Retorna o número da Porta.

7.4. Estado: Retorna o estado da porta, 'open' (aberta) ou 'closed' (fechada).

Conclusão

Muitas dificuldades surgiram, dentre elas a limpeza da tela e posteriormente a exibição dinâmica do menu após resposta da requisição. A principal dificuldade foi com o nmap, onde problemas inesperados aconteceram na parte de execução dos testes. Porém obtivemos pontos positivos, como a experiência adquirida ao lidar com tais situações, onde foi necessário pensar em uma outra abordagem para resolver o erro que apresentava no compilador Thonny. O segundo ponto foi o aprendizado que cada integrante obteve ao longo do processo. Para finalizar, o terceiro ponto foi a qualidade dos resultados obtidos que pode manter a essência do projeto de uma aplicação simples de monitoramento de recursos de um sistema operacional.

Código Cliente

```
import cpuinfo, datetime, time, platform, os, nmap, subprocess,
socket, pickle

cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

HOST = socket.gethostname()
PORT = 12345
HP = (HOST, PORT)
quant = 32768

def menu():

    print('=====')
    print('1- Processador')
    print('2- Memória')
    print('3- Armazenamento')
    print('4- Arquivos e diretórios')
    print('5- Processos | PID')
    print('6- Rede')
    print('7- Nmap')
    print('=====')

try:
    cliente.connect(HP)
    req = " "
    cliente.send(req.encode('utf-8'))

    while True:
        menu()
        req = input('Digite a opção desejada:')

        if (req == "1") | (req == "2") | (req == "3") | (req == "4")
| (req == "5") | (req == "6"): #| (req == "7"):
            cliente.send(req.encode('utf-8'))
            bytes_resp = cliente.recv(quant)
            resposta = pickle.loads(bytes_resp)

            if req == "1":
                print("\n1- Processador:\n")
                legenda_processador = ['Modelo:', 'Arquitetura:',
'Palavra (bits):', 'Frequência Base:', 'Frequência média (GHz):',
'Uso (%):', 'Uso núcleos (%):', 'Núcleos Lógicos:', 'Núcleos
```

```

Físicos:', 'Troca de contexto (boot):', 'Interrupções (boot):',
'Chamadas de sistema (boot):']
        var=' | '.join(map(str, resposta[6]))
        valor_processador = [resposta[0], resposta[1],
resposta[2], resposta[3], resposta[4], resposta[5], var, resposta[7],
resposta[8]]
        for x,y in
zip(legenda_processador, valor_processador):
            print ('{:30}{:30}'.format(x,y))
            time.sleep(10)
            os.system('cls')

    elif req == "2":
        print("\n2- Memória\n")
        legenda_memoria = ['RAM total (GB):', 'RAM livre
(GB):', 'RAM uso (%):', 'Swap total (GB):', 'Swap livre (GB):']
        for x,y in zip(legenda_memoria, resposta):
            print ('{:30}{:30}'.format(x,y))
            time.sleep(10)
            os.system('cls')

    elif req == "3":
        print("\n3- Armazenamento\n")
        legenda_armazenamento = ['Total (GB):', 'Livre
(GB):', 'Utilizado (%):']
        for x,y in zip(legenda_armazenamento, resposta):
            print ('{:30}{:30}'.format(x,y))
            time.sleep(10)
            os.system('cls')

    elif req == "4":

        print("\n4- Arquivos e Diretórios\n")
        print("")
        print("\nDiretórios\n")

        for diretorio in resposta[0]:
            print(diretorio)

        print("")
        print("\nArquivos\n")
        d1 = resposta[1]
        for key, d1 in d1.items():
            print(key)
            for attribute, value in d1.items():

```

```

        print('{}: {}'.format(attribute,value))
    print("")

    time.sleep(10)
    os.system('cls')

    elif req == "5":
        print("\n5- Processos | PID\n")
        legenda_pid = ['Nome:', 'PID:', 'Executável:', 'CPU
(s):', 'Memória (MB):']

        i = 0

        while i <= len(resposta):
            valor_pid = [resposta[i][0], resposta[i][1],
resposta[i][2], resposta[i][3], resposta[i][4]]
            i = i + 1

            for x,y in zip(legenda_pid,valor_pid):
                print ('{:30}{:<30}'.format(x,y))
            print("")
            time.sleep(10)
            os.system('cls')

    elif req == "6":
        print("\n6- Rede\n")
        legenda_rede = ['IPv4:', 'IPv6:', 'Máscara:', 'MAC:']
        valor_rede = [resposta[0], resposta[1], resposta[2],
resposta[3]]

        for x,y in zip(legenda_rede,valor_rede):
            print ('{:30}{:30}'.format(x,y))
        time.sleep(10)
        os.system('cls')

    if req == "7":
        endereco_ip = input("Entre com o número do IP da rede: ")
        cliente.send(req.encode('utf-8'))
        cliente.send(endereco_ip.encode('utf-8'))

        bytes_resp = cliente.recv(quant)
        resposta = pickle.loads(bytes_resp)

        print("\n7- NMAP\n")

        d1=resposta

```

```

        for key, d1 in d1.items():
            print(key)
            for attribute, value in d1.items():
                print('{}: {}'.format(attribute, value))
            print("")

    time.sleep(15)
    os.system('cls')

    legenda_nmap = ['IP (sub-rede):', 'Protocolo:', 'Porta:',
'Estado:']

        for key, value in resposta.items():
            valor_nmap = [resposta[key], resposta.get(key,
value[0]), resposta.get(key, value[1]), resposta.get(key, value[3])]

except Exception as erro:
    print(str(erro))

cliente.close()

input("Para sair pressione qualquer tecla...")

```

Código Servidor

```
import psutil, cpuinfo, datetime, time, platform, os, nmap,
subprocess, socket, pickle

servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

HOST = socket.gethostname()
PORT = 12345
HP = (HOST, PORT)
quant = 32768
nm = nmap.PortScanner()

servidor.bind(HP)
servidor.listen()
print("Servidor:", HOST, "esperando na porta", PORT, "*Beep...
*beep... *beep!")

(cliente, addr) = servidor.accept()
print("Máquina do cliente conectada:", addr)

def processador():

    inicio = time.time()

    lista = []
    lista_med = []

    info = cpuinfo.get_cpu_info()
    dic_cpu = []
    dic_cpu = psutil.cpu_stats()
    dic_cpu_nucleo = []
    dic_cpu_nucleo = psutil.cpu_percent(interval=1, percpu=True)
    lista_nucleo = []

    nome_cpu = info['brand']
    arq_cpu = info['arch']
    palavra_cpu = str(info['bits'])
    freq_base = info['hz_advertised']

    for i in range(10):
        freq_cpu = psutil.cpu_freq().current / 1000
        time.sleep(1)
        lista_med.append(freq_cpu)
```

```

freq_cpu = str(int(sum(lista_med)) / int(len(lista_med)))

lista_med_uso = []

for i in range(10):
    uso_cpu = psutil.cpu_percent()
    time.sleep(1)
    lista_med_uso.append(uso_cpu)

uso_cpu = str(int(sum(lista_med_uso)) / int(len(lista_med_uso)))
lista_nucleo = psutil.cpu_percent(interval=1, percpu=True)
nucleos_cpu_logicos = str(psutil.cpu_count(logical=True))
nucleos_cpu_fisicos = str(psutil.cpu_count(logical=False))

lista.append(nome_cpu)
lista.append(arq_cpu)
lista.append(palavra_cpu)

lista.append(freq_base)

lista.append(freq_cpu)
lista.append(uso_cpu)
lista.append(lista_nucleo)
lista.append(nucleos_cpu_logicos)
lista.append(nucleos_cpu_fisicos)

fim = time.time()
print("Tempo da chamada:", round((fim-inicio),2), "segundo(s).")

return lista

def memoria():

    inicio = time.time()

    lista = []

    lista_med = []

    memoria_total = round(psutil.virtual_memory().total/1024**3)
    memoria_livre = round(psutil.virtual_memory().free/1024**3)

    memoria_swap_total = round(psutil.swap_memory().total/1024**3)
    memoria_swap_livre = round(psutil.swap_memory().free/1024**3)

```

```

for i in range(10):
    memoria_uso = psutil.virtual_memory().percent
    time.sleep(1)
    lista_med.append(memoria_uso)

memoria_uso = round((sum(lista_med) / len(lista_med)))

lista.append(memoria_total)
lista.append(memoria_livre)
lista.append(memoria_uso)
lista.append(memoria_swap_total)
lista.append(memoria_swap_livre)

fim = time.time()
print("Tempo da chamada:", round((fim-inicio),2), "segundo(s).")

return lista

def armazenamento():

    inicio = time.time()

    lista = []

    armazenamento_total =
round(psutil.disk_usage(os.getcwd()).total/1024**3)
    armazenamento_livre =
round(psutil.disk_usage(os.getcwd()).free/1024**3)
    armazenamento_uso = round(psutil.disk_usage(os.getcwd()).percent)

    lista.append(armazenamento_total)
    lista.append(armazenamento_livre)
    lista.append(armazenamento_uso)

    fim = time.time()
    print("Tempo da chamada:", round((fim-inicio),2), "segundo(s).")

    return lista

def arquivo():

    inicio = time.time()

    diretorio = os.listdir()

```



```

dicionario_arq = {}
lista_diretorios = []
dicionario_arq_dados = {}

d2 = {}

for arquivo in diretorio:

    if os.path.isfile(arquivo):
        size=(round(os.stat(arquivo).st_size / 1024)) # tamanho
em bytes
        path=os.path.abspath(arquivo) # caminho absoluto
        filec=time.ctime(os.path.getctime(arquivo)) # criação do
arquivo
        filem=time.ctime(os.path.getmtime(arquivo)) # última
modificação
        filea=time.ctime(os.path.getatime(arquivo)) # último
acesso
        d2={'Tamanho (bytes)':size, 'Caminho
absoluto':path, 'Criação':filec, 'Última modificação':filem, 'Último
acesso':filea}
        dicionario_arq_dados.update({arquivo:d2})

    else:
        lista_diretorios.append(arquivo)

lista = [lista_diretorios, dicionario_arq_dados]

return lista

def pid():

    inicio = time.time()

    dicionario = {}
    key = 0

    for proc in psutil.process_iter():
        if psutil.pid_exists(proc.pid):
            processID = proc.pid

            try:

                processName = proc.name()
                processExecutable = proc.exe()

```

```

        processCpuTime = round(proc.cpu_times().user,2)
        processMemory = round((proc.memory_info().rss /
(1024*1024)),2)

        processCreateTime =
datetime.datetime.fromtimestamp(proc.create_time()).strftime("%d-%m-
%Y %H:%M:%S")

        lista = []
        lista.append(processName)
        lista.append(processID)
        lista.append(processExecutable)
        lista.append(processCpuTime)
        lista.append(processMemory)
        lista.append(processCreateTime)

        dicionario [key] = lista
        key = key + 1

    except (psutil.NoSuchProcess, psutil.AccessDenied,
psutil.ZombieProcess):
        pass

    fim = time.time()
    print("Tempo da chamada:", round((fim-inicio),2), "segundo(s).")

    return dicionario

def rede():

    inicio = time.time()

    interfaces = psutil.net_if_addrs()
    nomes = []
    status = psutil.net_if_stats()

    for i in interfaces:
        nomes.append(str(i))
    dic_itf = psutil.net_if_addrs()
    dic_nic = psutil.net_if_stats()
    dic_io = psutil.net_io_counters(pernic=True, nowrap=True)

    lista = []
    ipv4 = dic_itf[nomes[0]][1][1]
    ipv6 = dic_itf[nomes[0]][2][1]

```

```

mascara = dic_itf[nomes[0]][1][2]
mac = dic_itf[nomes[0]][0][1]

lista.append(ipv4)
lista.append(ipv6)
lista.append(mascara)
lista.append(mac)

fim = time.time()
print("Tempo da chamada:", round((fim-inicio),2), "segundo(s).")

return lista

def nmap():

    inicio = time.time()

    octeto = "0"
    dicionario = {}
    d2 = {}
    key = 0
    i = 0

    for octeto in range(6): #256

        lista = []
        ip = ("%s%s").strip() % (base_ipv4, octeto)

        nm.scan(ip)

        for host in nm.all_hosts():
            ip_subrede = str(host).strip()

            for proto in nm[host].all_protocols():
                protocolo_subrede = str(proto).upper()

                lport = nm[host][proto].keys()
                for port in lport:
                    porta_subrede = str(port).strip()
                    estado_porta_subrede =
str(nm[host][proto][port]['state']).strip()

                    d2={'Protocolo':protocolo_subrede,
'Porta':porta_subrede, 'Estado':estado_porta_subrede}

```

```

        dicionario.update({ip_subrede:d2})

fim = time.time()
print("Tempo da chamada:", round((fim-inicio),2), "segundo(s).")

return dicionario

while True:
    req = cliente.recv(quant)

    if req.decode('utf-8') == " ":
        print("Conexão bem-sucedida.")

    elif req.decode('utf-8') == "1":
        bytes_resp = pickle.dumps(processador())
        cliente.send(bytes_resp)

    elif req.decode('utf-8') == "2":
        bytes_resp = pickle.dumps(memoria())
        cliente.send(bytes_resp)

    elif req.decode('utf-8') == "3":
        bytes_resp = pickle.dumps(armazenamento())
        cliente.send(bytes_resp)

    elif req.decode('utf-8') == "4":
        bytes_resp = pickle.dumps(arquivo())
        cliente.send(bytes_resp)

    elif req.decode('utf-8') == "5":
        bytes_resp = pickle.dumps(pid())
        cliente.send(bytes_resp)

    elif req.decode('utf-8') == "6":
        bytes_resp = pickle.dumps(rede())
        cliente.send(bytes_resp)

    elif req.decode('utf-8') == "7":
        ip = cliente.recv(quant)
        endereco_ip = ip.decode('utf-8')
        base_ipv4 = endereco_ip

        nmap_resultado = nmap()
        bytes_resp = pickle.dumps(nmap_resultado)
        cliente.send(bytes_resp)

```

```
cliente.close()
servidor.close()

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

Referências

1. <https://psutil.readthedocs.io/en/latest/>
2. <https://github.com/workhorsy/py-cpuinfo>
3. <https://xael.org/norman/python/python-nmap/>
4. https://nmap.org/man/pt_BR/

Glossário

Bit(s):

Menor unidade de informação que pode ser armazenada ou transmitida.

Byte(s):

Conjunto estruturado de dados formado por uma sequência ordenada de oito bits.

Bytes escritos:

Quantidade de bytes que são escritos em disco em determinado intervalo de tempo.

Bytes lidos:

Quantidade de bytes que são lidos em disco em determinado intervalo de tempo.

Cliente:

Máquina com perfil ativo que envia solicitações para realização de processos em um sistema computacional com perfil passivo (servidor).

Cliente-Servidor:

Arquitetura baseada em um computador (cliente) que envia requisições para serem processadas em outro computador (servidor) e espera receber uma resposta de volta. Geralmente referenciada como um computador pessoal que envia requisições para um conjunto computacional processá-las e retorná-las como resposta para o computador de origem.

CMD:

Interpretador de linha de comando utilizado no sistema operacional Windows.

Compilador:

Programa de computador que a partir de um código em determinada linguagem de programação compilada gera um programa semanticamente equivalente, porém em outra linguagem, código objeto.

Dicionário:

Estrutura de armazenamento da linguagem Python, baseada em uma chave referenciando a um item e um valor que pode ser unitário ou outra estrutura de armazenamento (outro dicionário, lista, tupla).

DPC (Deferred Procedure Call):

Mecanismo do Sistema operacional Windows que permite que tarefas de alta-prioridade adiem tarefas necessárias com prioridade baixa para execução posterior.

Giga-hertz:

Equivale a 1.000.000.000 hertz.

I/O:

Da computação Input/Output. Entrada e saída de dados.

Interrupções de hardware:

Sinal resultante da troca de contexto de um dispositivo.

IP:

Número atribuído a um computador quando conectado a uma rede.

Kilobyte:

Equivalente a 1.000 bytes.

Lista:

Estrutura de armazenamento da linguagem Python, baseada em posições onde cada posição pode ser apenas um valor unitário ou outra estrutura de armazenamento (outra lista, dicionário, tupla).

NIC (Network Interface Card – “Placa de Rede”):

Hardware presente em um sistema computacional que é responsável pela comunicação entre o computador e uma rede de computadores (E.g.: Internet)

Nmap:

Software que realiza escaneamento de portas de um computador.

Porta:

Software de aplicação ou processo que serve de ponto final de comunicação de um serviço hospedeiro.

Power Shell:

Ferramenta multiplataforma disponível no Windows e voltada para automação e configuração do sistema por meio de scripts (códigos).

Processador:

Circuito integrado que realiza as funções de cálculo e tomada de decisão de um computador.

Sub-rede:

Subdivisão lógica de uma rede de computadores.

TCP (Transmission Control Protocol):

Conjunto de protocolos de comunicação entre computadores em rede.

UDP (User Datagram Protocol):

Protocolo de transferência de dados da camada de transporte.

Windows:

Formalmente chamado de Microsoft Windows. Sistema operacional desenvolvido pela empresa norte-americana Microsoft Corporation.