

INSTITUTO INFNET
ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO



Teste de Performance 9

Projeto de Bloco

Marcelo da Silva Oliveira

Prof.: Alcione Dolavale

Rio de Janeiro, 2021

Relatório

Um aplicativo simples de apresentação gráfica do monitoramento e análise do computador. Ele foi implementado em Python usando módulos como psutil (para capturar dados do sistema computacional) e Pygame (para exibir graficamente os dados).

Três bibliotecas do Python foram utilizadas para o desenvolvimento da aplicação, sendo elas *psutil*, *pygame*, *platform* e *cpuinfo*.

Psutil

É uma biblioteca de plataforma cruzada para recuperar informações sobre os processos em execução e a utilização do sistema (CPU, memória, discos, rede, sensores) em Python. É útil principalmente para monitoramento de sistema, criação de perfil, limitação de recursos de processo e gerenciamento de processos em execução.

Pygame

Pygame é um conjunto de módulos Python projetados para escrever videogames. Isso permite que se crie jogos completos e programas multimídia na linguagem python. Foi utilizada nesta aplicação para exibição das informações recuperadas do sistema.

Platform

Biblioteca utilizada para buscar informações sobre a arquitetura do processador, como o nome e modelo. Além disso, também é possível encontrar informações sobre o sistema operacional.

CPUINFO

Biblioteca utilizada para buscar algumas informações sobre o processador.

Primeiramente foi criado uma interface gráfica para mostrar o uso da memória, da CPU com os dados do processador, do uso de disco e IP do computador.

Eles serão apresentados como uma barra indicativa de quanto está em uso e quanto está livre, exceto o IP do computador e os dados do processador. Estes serão apresentados apenas com textos.

Para criar a interface gráfica, inicialmente foi gerado uma janela e uma estrutura de repetição até que o botão de sair seja pressionado.

```
import pygame, psutil, platform

# Iniciando a janela principal
largura_tela = 800
altura_tela = 690
tela = pygame.display.set_mode((largura_tela, altura_tela))
pygame.display.set_caption("Monitoramento e análise do computador")
pygame.display.init()
vermelho = (255,0,0)
azul = (0,0,255)
verde = (0,255,0)
preto = (0,0,0)
branco = (255,255,255)
s1 = pygame.surface.Surface((largura_tela, altura_tela/3))
s2 = pygame.surface.Surface((largura_tela, altura_tela/3))
s3 = pygame.surface.Surface((largura_tela, altura_tela/3))
```

Criou-se uma janela de largura 800 e altura 690 com o título “Monitoramento e análise do computador”. Além disso, definimos as cores vermelho, azul, preto, branco e verde para utilizarmos nos próximos passos e utilizamos o *pygame surface* para criar sub janelas dentro da janela principal para colocar as análises do computador.

Barra de Uso da memória

Após a criação da janela foi criado uma função chamada “mostra_uso_memoria” para mostrar um gráfico com o uso da memória do computador.

```
pygame.font.init()
font = pygame.font.Font(None, 22)

def mostra_uso_memoria():
    mem = psutil.virtual_memory()
    perc = mem.percent
    larg = largura_tela - 2*20
    tela.fill(preto)
    pygame.draw.rect(s1, azul, (20, 60, larg, 70))
    tela.blit(s1, (0, 0))
    larg = larg*mem.percent/100
    pygame.draw.rect(s1, vermelho, (20, 60, larg, 70))
    tela.blit(s1, (0, 0))
    total = round(mem.total/(1024*1024*1024),2)
    texto_barra = "Uso de Memória (Total: " + str(total) + "GB): " + str(perc) + "%"
    text = font.render(texto_barra, 1, branco)
    tela.blit(text, (20, 10))
```

Essa função mostra o valor da memória total, com a porcentagem escrita e um gráfico com retângulos vermelho e azul para descrever a memória utilizada e a memória livre, respectivamente.

Note que as informações da memória foram captadas através da função “psutil.virtual_memory()”.

A exibição da barra de uso da memória ficou da seguinte forma:



Barra de uso detalhado da CPU e informação detalhada da plataforma de processamento

Após a criação da barra de uso da memória, foi utilizado a mesma estrutura de código para criar a barra de uso detalhado da CPU, detalhando a porcentagem de uso de cada core(núcleo). Além disso, foi adicionado também informações mais detalhadas da CPU como, o nome do processador, arquitetura e a palavra(bits) através do “cpuinfo”. As informações de núcleos físicos e lógicos e a frequência são retiradas do “psutil”.

Os núcleos do processador são divididos em núcleos físicos e lógicos. Os núcleos físicos são hardwares que estão dentro da CPU. Os núcleos lógicos são as habilidades de um núcleo físico fazer duas ou mais atividades simultaneamente.

Os processadores podem possuir arquitetura CISC e RISC. Processador de arquitetura CISC executa uma instruções mais complexas. Com isso, o tempo de processamento será maior afetando, portanto, a capacidade de processamento. Para executar uma só instrução, um processador CISC pode exigir vários ciclos de relógio. Já os processadores de arquitetura RISC executam instruções reduzidas, ou seja, ele quebra a instrução em várias menores e mais simples e todas assumem um tamanho padrão. Cada uma dessas instruções têm as características necessárias para que possa ser executada em apenas um ciclo de relógio.

Além da arquitetura, o “cpuinfo” também indica o tamanho da palavra. A palavra é utilizada para indicar a unidade de transferência e processamento de um computador. As palavras são múltiplos de 1 byte, sendo que os microprocessadores geralmente utilizam 32bits – 4 bytes como tamanho da palavra (já existem projetos e

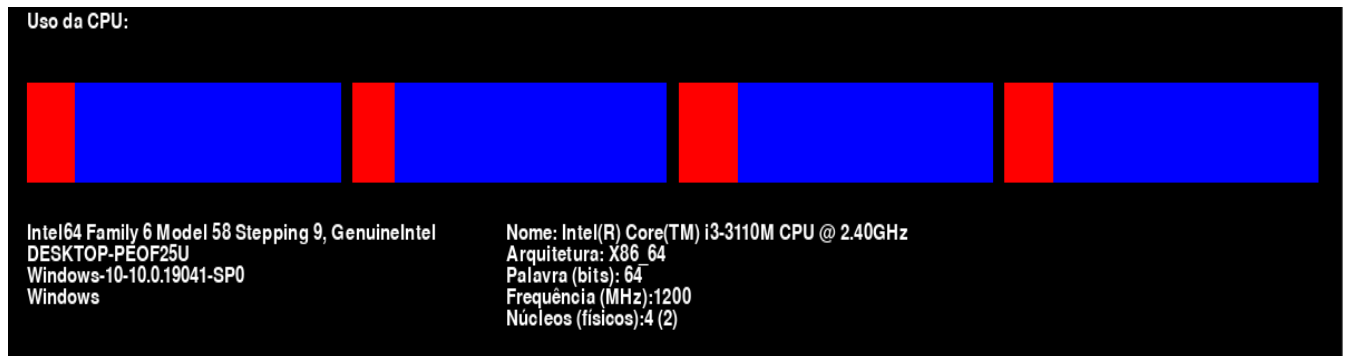
microprocessadores que utilizam palavras de 64 bits, porém estes microprocessadores ainda não se popularizaram).

Para capturar a porcentagem de uso de processamento de cada núcleo do processador, foi usado a mesma função utilizada na TP2(`psutil.cpu_percent`), mas adicionando um parâmetro a ela.

```
def mostra_uso_cpu(s, l_cpu_percent):
    num_cpu = len(l_cpu_percent)
    x = y = 10
    desl = 10
    larg = (s.get_width()-2*y - (num_cpu+1)*desl)/num_cpu
    d = x + desl
    for i in l_cpu_percent:
        pygame.draw.rect(s, azul, (d, y, larg, 70))
        pygame.draw.rect(s, vermelho, (d, y, (i/100)*larg, 70))
        d = d + larg + desl
    tela.blit(s3, (0, 6*altura_tela/10))
    texto_barra_2 = "Uso da CPU:"
    text2 = font.render(texto_barra_2, 1, branco)
    tela.blit(text2, (20, 6*altura_tela/10 - 40))
    texto_proc = str(platform.processor())
    textp = font.render(texto_proc, 1, branco)
    tela.blit(textp, (20, 7*altura_tela/10 + 40))
    texto_node = str(platform.node())
    textn = font.render(texto_node, 1, branco)
    tela.blit(textn, (20, 7*altura_tela/10 + 55))
    texto_plat = str(platform.platform())
    textpl = font.render(texto_plat, 1, branco)
    tela.blit(textpl, (20, 7*altura_tela/10 + 70))
    texto_sistema = str(platform.system())
    texts = font.render(texto_sistema, 1, branco)
    tela.blit(texts, (20, 7*altura_tela/10 + 85))
    texto_nome = "Nome: " + str(info_cpu["brand_raw"])
    textnm = font.render(texto_nome, 1, branco)
    tela.blit(textnm, (450, 7*altura_tela/10 + 40))
    texto_arq = "Arquitetura: " + str(info_cpu["arch"])
    textar = font.render(texto_arq, 1, branco)
    tela.blit(textar, (450, 7*altura_tela/10 + 55))
    texto_palavra = "Palavra (bits): " + str(info_cpu["bits"])
    textpa = font.render(texto_palavra, 1, branco)
    tela.blit(textpa, (450, 7*altura_tela/10 + 70))
    texto_freq = "Frequência (MHz):" + str(round(psutil.cpu_freq().current))
    textfr = font.render(texto_freq, 1, branco)
    tela.blit(textfr, (450, 7*altura_tela/10 + 85))
```

Note que, a estrutura de código da função “mostra_uso_CPU” foram colocados informações detalhadas do processamento através da biblioteca “platform”, foram colocados, também, informações detalhadas da CPU através das bibliotecas “cpuinfo”(nome, arquitetura e palavra) e “psutil”(frequência, núcleos lógicos e físicos).

A barra de uso da CPU com a informação detalhada do processador ficou da seguinte forma:



Barra de uso de Disco

Para criar a barra de uso de Disco, também foi utilizado a mesma estrutura de código do uso da memória e da CPU, porém com as informações de uso do mesmo.

```
def mostra_uso_disco():
    disco = psutil.disk_usage('.')
    larg = largura_tela - 2*20
    pygame.draw.rect(s3, azul, (20, 50, larg, 70))
    tela.blit(s3, (0, 6*altura_tela/10 - 50))
    larg = larg*disco.percent/100
    pygame.draw.rect(s3, vermelho, (20, 50, larg, 70))
    tela.blit(s3, (0, 6*altura_tela/10 - 50))
    total = round(disco.total/(1024*1024*1024), 2)
    texto_barra_3 = "Uso de Disco: (Total: " + str(total) + "GB): " + str(disco.percent) + "%"
    text3 = font.render(texto_barra_3, 1, branco)
    tela.blit(text3, (20, 6*altura_tela/10 - 30))
```

Note que a estrutura de código do uso de Disco é parecido com os códigos do uso de memória e do uso da CPU, porém com as informações específicas do Disco utilizando a função “psutil.disk_usage('.')”.

A barra de uso do disco ficou da seguinte forma:

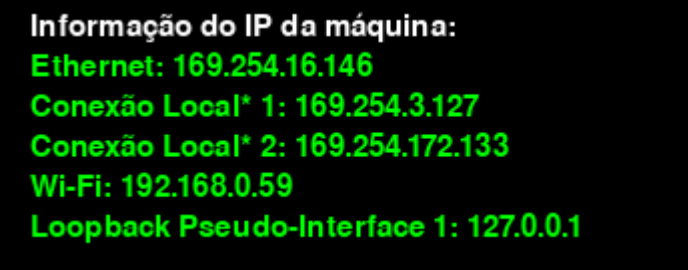


Informações do IP da máquina:

Para exibir as informações do IP da máquina, foi utilizado uma estrutura parecida com as das informações anteriores, porém não teve a utilização da barra. Além disso, utilizamos a função “psutil.net_if_addrs()” para exibir as informações do IP do computador.

```
def IP_maquina():
    dic_interfaces = psutil.net_if_addrs()
    texto_ip = 'Informação do IP da máquina:'
    textip = font.render(texto_ip, 1, branco)
    tela.blit(textip, (20, 7*altura_tela/10 + 20))
    texto_ET = 'Ethernet: ' + str(dic_interfaces['Ethernet'][1].address)
    textet = font.render(texto_ET, 1, verde)
    tela.blit(textet, (20, 7*altura_tela/10 + 40))
    texto_CL1 = 'Conexão Local* 1: ' + str(dic_interfaces['Conexão Local* 1'][1].address)
    textcl1 = font.render(texto_CL1, 1, verde)
    tela.blit(textcl1, (20, 7*altura_tela/10 + 60))
    texto_CL2 = 'Conexão Local* 2: ' + str(dic_interfaces['Conexão Local* 2'][1].address)
    textcl2 = font.render(texto_CL2, 1, verde)
    tela.blit(textcl2, (20, 7*altura_tela/10 + 80))
    texto_WF = 'Wi-Fi: ' + str(dic_interfaces['Wi-Fi'][1].address)
    textwf = font.render(texto_WF, 1, verde)
    tela.blit(textwf, (20, 7*altura_tela/10 + 100))
    texto_PI = 'Loopback Pseudo-Interface 1: ' + str(dic_interfaces['Loopback Pseudo-Interface 1'][0].address)
    textpi = font.render(texto_PI, 1, verde)
    tela.blit(textpi, (20, 7*altura_tela/10 + 120))
```

A exibição das informações do IP da máquina ficou da seguinte forma:



Informação do IP da máquina:
Ethernet: 169.254.16.146
Conexão Local* 1: 169.254.3.127
Conexão Local* 2: 169.254.172.133
Wi-Fi: 192.168.0.59
Loopback Pseudo-Interface 1: 127.0.0.1

Arquivos e diretórios

A partir dessa fase será utilizado a o módulo 'OS' para buscar informações sobre arquivos e diretórios como o nome, data de criação, data de modificação e tamanho. A primeira estrutura de código utilizada imprime os dados de todos os arquivos que estão dentro do diretório.

```
import os, time
# Obtém lista de arquivos e diretórios do diretório corrente:
lista = os.listdir()
dic = {} # cria dicionário
for i in lista: # Varia na lista dos arquivos e diretórios
    if os.path.isfile(i): # checa se é um arquivo
        # Cria uma lista para cada arquivo. Esta lista contém o
        # tamanho, data de criação e data de modificação.
        dic[i] = []
        dic[i].append(os.stat(i).st_size) # Tamanho
        dic[i].append(os.stat(i).st_atime) # Tempo de criação
        dic[i].append(os.stat(i).st_mtime) # Tempo de modificação

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(titulo)
for i in dic:
    kb = dic[i][0]/1024
    tamanho = '{:10}'.format(str('{:.2f}'.format(kb)+' KB'))
    print(tamanho, time.ctime(dic[i][2]), " ", time.ctime(dic[i][1]), " ", i)
```

Foram utilizados as funções “os.stat()”, “os.listdir()” e “os.isfile()”. Além dessas funções, também foi usado uma formatação do python pra poder melhorar o alinhamento e deixar o resultado mais organizado e foi utilizado o módulo ‘time’ para poder colocar a data e hora da forma que utilizamos.

Tamanho	Data de Modificação	Data de Criação	Nome
1.06 KB	Tue Nov 16 23:35:22 2021	Tue Nov 16 23:35:23 2021	aula 2.txt
1.06 KB	Tue Nov 16 23:35:31 2021	Tue Nov 16 23:35:32 2021	aula.txt
1.90 KB	Thu Sep 23 07:15:32 2021	Fri Nov 12 12:07:08 2021	Botão.py
37.35 KB	Thu Sep 23 07:21:42 2021	Thu Nov 18 08:46:44 2021	Capa TP_AT - 2021.docx
0.02 KB	Fri Sep 10 07:51:43 2021	Fri Nov 12 12:07:08 2021	conta.py
0.34 KB	Fri Aug 13 08:11:05 2021	Fri Nov 12 12:07:08 2021	CPU INFO.py
1.89 KB	Fri Sep 10 11:50:57 2021	Fri Nov 12 12:07:08 2021	CPU pygame.py
0.19 KB	Fri Jul 23 10:55:13 2021	Fri Nov 12 12:07:08 2021	CUMSUM.py
0.67 KB	Thu Nov 18 08:44:17 2021	Mon Nov 22 16:27:21 2021	Etapa 4 Ex 7.py
1.17 KB	Fri Nov 19 01:32:02 2021	Mon Nov 22 16:25:27 2021	Etapa 4 Ex 10 PB.py
0.40 KB	Fri Nov 12 11:26:42 2021	Tue Nov 16 23:07:30 2021	Etapa 4 Ex 3 PB.py
0.69 KB	Tue Nov 16 23:10:51 2021	Fri Nov 19 01:35:27 2021	Etapa 4 Ex 4 PB.py
0.36 KB	Thu Nov 18 20:08:37 2021	Fri Nov 19 01:36:42 2021	Etapa 4 Ex 8 PB.py
0.64 KB	Fri Nov 19 01:14:23 2021	Fri Nov 19 01:34:40 2021	Etapa 4 Ex 9 PB.py
0.08 KB	Thu Aug 26 12:11:06 2021	Fri Nov 12 12:07:08 2021	IP.py
3.36 KB	Tue Aug 17 09:49:15 2021	Fri Nov 12 12:07:08 2021	Jogo do quadradinho.py
239.70 KB	Mon Sep 20 07:11:12 2021	Fri Nov 12 12:07:08 2021	Marcelo_Oliveira_DR1_AT.pdf
0.65 KB	Thu Aug 26 11:33:49 2021	Fri Nov 12 12:07:08 2021	Memória - 1.py

Outra estrutura de código foi utilizada para calcular o tamanho do diretório cujo nome foi enviado pelo usuário. As funções utilizadas para esse código foi 'os.listdir()', 'os.path.isdir()', 'os.path.isfile()', 'os.path.join()' e 'os.stat()'.

```
1 import os
2 lista_dir = []
3 p_dir = ''
4 entrada = input('Entre com o diretório: ')
5 if os.path.isdir(entrada):
6     lista_dir.append(entrada)
7     somador = 0
8     while lista_dir:
9         diretorio = lista_dir[0]
10        p_dir = os.path.join(p_dir, diretorio)
11        lista = os.listdir(p_dir)
12        for i in lista:
13            p = os.path.join(p_dir, i)
14            if os.path.isdir(p):
15                lista_dir.append(i)
16            elif os.path.isfile(p):
17                somador = somador + os.stat(p).st_size
18        lista_dir.remove(diretorio)
19    print(str(round(somador/1024, 2)) + ' KB')
20 else:
21    print("O diretório", '\\' + entrada + '\\', 'não existe.')
22
```

Shell x

```
>>> %Run 'Estapa 4 Ex 7.py'
```

```
Entre com o diretório: C:\Users\win10\Thonny
319.09 KB
```

Informações de Processo

Para listar as informações de processo é necessário a utilização dos módulos 'subprocess', 'psutil' e 'time'.

Foi criada uma estrutura de código para listar as informações de execução de um processo que o usuário envia. As informações são: nome, PID, executável, tempo de uso de CPU do usuário e do sistema, percentual de uso de CPU, uso de memória, percentual do uso de memória principal e número de threads pertencentes ao processo.

Para identificar o processo, foi utilizado o PID e para acessar as informações do processo, foi utilizado "psutil.Process(PID)". O módulo "subprocess" foi utilizado para que o próprio processo criasse um outro processo que foi aquele monitorado.

```

1 import subprocess, psutil, time
2
3 nome = input('Idinque o nome do processo :')
4 pid = subprocess.Popen(nome).pid
5
6 p = psutil.Process(pid)
7 print('Nome: ', p.name())
8 print('Executável: ', p.exe())
9 print('Tempo de criação: ', time.ctime(p.create_time()))
10 print('Tempo de usuário: ', p.cpu_times().user, 's')
11 print('Tempo de sistema: ', p.cpu_times().system, 's')
12 print('Percentual de uso de CPU:: ', p.cpu_percent(interval = 1.0), '%')
13 perc_mem = '{:.2f}'.format(p.memory_percent())
14 print('Percentual de uso de memória: ', perc_mem, '%')
15 mem = '{:.2f}'.format(p.memory_info().rss/1024/1024)
16 print('Uso de memória: ', mem, 'MB')
17 print('Número de threads: ', p.num_threads())
18

```

```

hell x
>> %Run "Etapa 4 Ex 9 PB.py"

Idinque o nome do processo :notepad.exe
Nome: notepad.exe
Executável: C:\Windows\SysWOW64\notepad.exe
Tempo de criação: Mon Nov 22 18:13:41 2021
Tempo de usuário: 0.0 s
Tempo de sistema: 0.0 s
Percentual de uso de CPU:: 20.3 %
Percentual de uso de memória: 0.23 %
Uso de memória: 13.75 MB
Número de threads: 7

```

Outra estrutura de código foi feito pra imprimir a lista de informações de todos os processos da máquina.

Como o usuário comum pode não ter permissão suficiente para obter todas as informações de determinado processo, foi feito uma verificação na qual o computador realizou uma ação somente se não tivesse ocorrido nenhum erro. Para isso, utilizou-se o artifício Python try-except.

```

import subprocess, psutil, time

def mostra_info(pid):
    try:
        p = psutil.Process(pid)
        texto = '{:6}'.format(pid)
        texto = texto + '{:11}'.format(p.num_threads())
        texto = texto + " " + time.ctime(p.create_time()) + " "
        texto = texto + '{:8.2f}'.format(p.cpu_times().user)
        texto = texto + '{:8.2f}'.format(p.cpu_times().system)
        texto = texto + '{:10.2f}'.format(p.memory_percent()) + " %"
        rss = p.memory_info().rss/1024/1024
        texto = texto + '{:10.2f}'.format(rss) + " MB"
        vms = p.memory_info().vms/1024/1024
        texto = texto + '{:10.2f}'.format(vms) + " MB"
        texto = texto + " " + p.exe()
        print(texto)
    except:
        pass

titulo = '{:^7}'.format("PID")
titulo = titulo + '{:^11}'.format("# THREADS")
titulo = titulo + '{:^26}'.format("CRIAÇÃO")
titulo = titulo + '{:^9}'.format("T. USU.")
titulo = titulo + '{:^9}'.format("T. SIS.")
titulo = titulo + '{:^12}'.format("MEM. (%)")
titulo = titulo + '{:^12}'.format("RSS")
titulo = titulo + '{:^12}'.format("VMS")
titulo = titulo + " EXECUTÁVEL"
print(titulo)
lista = psutil.pids()
for i in lista:
    mostra_info(i)

```

PID	#	THREADS		CRIAÇÃO		T. USU.	T. SIS.	MEM. (%)	RSS	VMS	EXECUTÁVEL
4	174	Wed Dec 31 21:00:00 1969		0.00	5705.75	0.15	%	9.04 MB	0.23 MB		
68	8	Wed Nov 10 00:25:27 2021		5.14	6.36	0.18	%	10.71 MB	8.83 MB	C:\Windows\System32\taskhostw.exe	
100	4	Wed Nov 10 00:21:43 2021		0.00	10.20	0.46	%	27.24 MB	6.20 MB	Registry	
312	27	Wed Nov 10 00:23:27 2021		180.77	93.98	1.45	%	86.19 MB	87.69 MB	C:\Program Files\Avast Software\	
356	14	Wed Nov 17 20:53:11 2021		0.12	0.38	0.03	%	1.51 MB	18.39 MB	C:\Program Files\WindowsApps\Mic	
396	22	Wed Nov 10 00:22:51 2021		57.08	114.33	0.43	%	25.39 MB	19.27 MB	C:\Windows\System32\svchost.exe	
488	5	Wed Nov 10 00:22:52 2021		1.42	0.75	0.14	%	8.08 MB	5.74 MB	C:\Windows\System32\fontdrvhost.exe	
520	2	Wed Nov 10 00:22:01 2021		0.00	0.23	0.01	%	0.33 MB	1.04 MB	C:\Windows\System32\smss.exe	
560	5	Wed Nov 10 00:22:52 2021		0.75	0.78	0.01	%	0.68 MB	1.77 MB	C:\Windows\System32\fontdrvhost.exe	
576	1	Wed Nov 10 00:26:50 2021		0.11	0.48	0.06	%	3.36 MB	2.25 MB	C:\Windows\System32\SecurityHeal	
660	15	Wed Nov 10 00:22:49 2021		2.73	11.53	0.04	%	2.34 MB	2.23 MB	C:\Windows\System32\csrss.exe	
752	1	Wed Nov 10 00:22:50 2021		0.03	0.09	0.02	%	1.30 MB	1.33 MB	C:\Windows\System32\wininit.exe	
764	13	Wed Nov 10 00:22:50 2021		2.56	127.81	0.05	%	2.95 MB	2.53 MB	C:\Windows\System32\csrss.exe	
804	13	Wed Nov 10 00:22:53 2021		172.36	118.25	0.27	%	15.86 MB	15.45 MB	C:\Windows\System32\svchost.exe	
832	7	Wed Nov 10 00:22:54 2021		8.16	11.25	0.07	%	3.93 MB	2.82 MB	C:\Windows\System32\svchost.exe	
836	9	Wed Nov 10 00:22:50 2021		29.83	56.12	0.11	%	6.78 MB	6.39 MB	C:\Windows\System32\services.exe	
844	9	Wed Nov 10 00:22:50 2021		47.50	38.78	0.23	%	13.64 MB	9.92 MB	C:\Windows\System32\lsass.exe	
932	6	Wed Nov 10 00:22:51 2021		0.30	1.38	0.08	%	4.49 MB	2.77 MB	C:\Windows\System32\winlogon.exe	
948	3	Wed Nov 10 00:25:52 2021		15.89	41.45	0.10	%	6.23 MB	3.05 MB	C:\Windows\System32\svchost.exe	
1044	12	Mon Nov 22 16:44:38 2021		4.16	13.36	0.24	%	14.06 MB	5.57 MB	C:\Windows\System32\svchost.exe	
1076	14	Wed Nov 10 00:22:54 2021		1741.39	960.02	0.93	%	55.65 MB	71.86 MB	C:\Windows\System32\dmw.exe	
1108	8	Wed Nov 10 00:22:57 2021		23.02	23.95	0.07	%	4.05 MB	3.62 MB	C:\Windows\System32\svchost.exe	

Monitoramento e Gerenciamento de processos

Nessa fase, foi utilizado o módulo 'sched' e 'time' para poder organizar, da melhor forma, a execução dos processos.

Sched

A biblioteca sched é utilizada para gerenciar a execução de processos, ou seja, utilizamos suas funções para escolher quando um processo vai ocorrer e quando vai ser interrompido.

Foi utilizado o módulo 'sched' para criar uma função que chamasse a função que lista as informações de um processo, que foram mencionados na fase anterior. Para isso, foi criada uma função chamada 'Long_event(name)' que executa a função que lista todas as informações dos arquivos e diretórios da fase anterior.

```
import sched, time, subprocess, psutil

scheduler = sched.scheduler(time.time, time.sleep)

def Long_event(name):
    print('EVENTO:', time.ctime(), time.process_time(), name)
    pid = subprocess.Popen('notepad.exe').pid

    p = psutil.Process(pid)
    print('Nome: ', p.name())
    print('Executável: ', p.exe())
    print('Tempo de criação: ', time.ctime(p.create_time()))
    print('Tempo de usuário: ', p.cpu_times().user, 's')
    print('Tempo de sistema: ', p.cpu_times().system, 's')
    print('Percentual de uso de CPU:: ', p.cpu_percent(interval = 1.0), '%')
    perc_mem = '{:.2f}'.format(p.memory_percent())
    print('Percentual de uso de memória: ', perc_mem, '%')
    mem = '{:.2f}'.format(p.memory_info().rss/1024/1024)
    print('Uso de memória: ', mem, 'MB')
    print('Número de threads: ', p.num_threads())
    print()
```

Após a criação dessa função, foi utilizado funções da biblioteca 'sched' pra poder chamar a função "Long_event(name)".

```
print('INICIO:', time.ctime())
scheduler.enterabs(1, 1, print_event, ('primeira chamada',))
print('CHAMADAS ESCALONADAS DA FUNÇÃO:', time.ctime(), time.process_time())

scheduler.run()
```

```
>>> %Run t2.py
```

```
INICIO: Wed Nov 24 20:09:36 2021
CHAMADAS ESCALONADAS DA FUNÇÃO: Wed Nov 24 20:09:36 2021 0.359375
EVENTO: Wed Nov 24 20:09:36 2021 0.359375 primeira chamada
Nome: notepad.exe
Executável: C:\Windows\SysWOW64\notepad.exe
Tempo de criação: Wed Nov 24 20:09:36 2021
Tempo de usuário: 0.0 s
Tempo de sistema: 0.0 s
Percentual de uso de CPU:: 18.5 %
Percentual de uso de memória: 0.23 %
Uso de memória: 13.96 MB
Número de threads: 7
```

Para chamar mais de uma função, foi criada uma função de escalonamento de chamadas através de prioridade e delay. Foram utilizados 3 eventos para escalonar: lista de informações de arquivos e diretórios, lista de informações de processos, e lista de informações de um processo, respectivamente.

```
1 def Long_event_1(name):
2     print ('INICIO DO EVENTO:', time.ctime(), time.process_time(), name)
3     lista = os.listdir()
4     dic = {} # cria dicionário
5     for i in lista: # Varia na lista dos arquivos e diretórios
6         if os.path.isfile(i): # checa se é um arquivo
7             # Cria uma lista para cada arquivo. Esta lista contém o
8             # tamanho, data de criação e data de modificação.
9             dic[i] = []
10            dic[i].append(os.stat(i).st_size) # Tamanho
11            dic[i].append(os.stat(i).st_atime) # Tempo de criação
12            dic[i].append(os.stat(i).st_mtime) # Tempo de modificação
13
14            titulo = '{:11}'.format("Tamanho") # 10 caracteres + 1 de espaço
15            # Concatenar com 25 caracteres + 2 de espaços
16            titulo = titulo + '{:27}'.format("Data de Modificação")
17            # Concatenar com 25 caracteres + 2 de espaços
18            titulo = titulo + '{:27}'.format("Data de Criação")
19            titulo = titulo + "Nome"
20            print(titulo)
21            for i in dic:
22                kb = dic[i][0]/1024
23                tamanho = '{:10}'.format(str('{:.2f}'.format(kb)+ ' KB'))
24                print(tamanho, time.ctime(dic[i][2]), " ", time.ctime(dic[i][1]), " ", i)
25            time.sleep(2)
26            print ('FIM DO EVENTO:', time.ctime(), time.process_time())
27            print()
```

```

1  def Long_event_2(name):
2      print ('INICIO DO EVENTO:', time.ctime(), time.process_time(), name)
3      def mostra_info(pid):
4          try:
5              p = psutil.Process(pid)
6              texto = '{:6}'.format(pid)
7              texto = texto + '{:11}'.format(p.num_threads())
8              texto = texto + " " + time.ctime(p.create_time()) + " "
9              texto = texto + '{:8.2f}'.format(p.cpu_times().user)
10             texto = texto + '{:8.2f}'.format(p.cpu_times().system)
11             texto = texto + '{:10.2f}'.format(p.memory_percent()) + " %"
12             rss = p.memory_info().rss/1024/1024
13             texto = texto + '{:10.2f}'.format(rss) + " MB"
14             vms = p.memory_info().vms/1024/1024
15             texto = texto + '{:10.2f}'.format(vms) + " MB"
16             texto = texto + " " + p.exe()
17             print(texto)
18         except:
19             pass
20         titulo = '{:^7}'.format("PID")
21         titulo = titulo + '{:^11}'.format("# THREADS")
22         titulo = titulo + '{:^26}'.format("CRIAÇÃO")
23         titulo = titulo + '{:^9}'.format("T. USU.")
24         titulo = titulo + '{:^9}'.format("T. SIS.")
25         titulo = titulo + '{:^12}'.format("MEM. (%)")
26         titulo = titulo + '{:^12}'.format("RSS")
27         titulo = titulo + '{:^12}'.format("VMS")
28         titulo = titulo + " EXECUTÁVEL"
29         print(titulo)
30         lista = psutil.pids()
31         for i in lista:
32             mostra_info(i)
33     print('FIM DO EVENTO:', time.ctime(), time.process_time())
34     print()

```

```

1  def Long_event_3(name):
2      print ('EVENTO:', time.ctime(), time.process_time(), name)
3      pid = subprocess.Popen('notepad.exe').pid
4
5      p = psutil.Process(pid)
6      print('Nome: ', p.name())
7      print('Executável: ', p.exe())
8      print('Tempo de criação: ', time.ctime(p.create_time()))
9      print('Tempo de usuário: ', p.cpu_times().user, 's')
10     print('Tempo de sistema: ', p.cpu_times().system, 's')
11     print('Percentual de uso de CPU: ', p.cpu_percent(interval = 1.0), '%')
12     perc_mem = '{:.2f}'.format(p.memory_percent())
13     print('Percentual de uso de memória: ', perc_mem, '%')
14     mem = '{:.2f}'.format(p.memory_info().rss/1024/1024)
15     print('Uso de memória: ', mem, 'MB')
16     print('Número de threads: ', p.num_threads())
17     print ('FIM DO EVENTO:', time.ctime(), time.process_time())
18     print()

```

Foi escolhido um delay menor para o “Long_event_1” em relação ao “Long_event_2” e “Long_event_3”, que possuem o mesmo delay. Porém, o “Long_event_1” tem prioridade 3, “Long_event_2” prioridade 2 e “Long_event_3” prioridade 1.

```

print('INICIO:', time.ctime())
scheduler.enter(2, 3, Long_event_1, ('LISTA DE INFORMAÇÕES DE ARQUIVOS E DIRETÓRIOS',))
scheduler.enter(8, 2, Long_event_2, ('LISTA DE INFORMAÇÕES DE PROCESSOS',))
scheduler.enter(8, 1, Long_event_3, ('LISTA DE INFORMAÇÕES DE UM PROCESSO',))
print('CHAMADAS ESCALONADAS DA FUNÇÃO:', time.ctime(), time.process_time())

scheduler.run()

```

Após executar a função, como o delay do “Long_event_1” é menor que os outros eventos, ele foi chamado primeiro. Após isso, o “Long_event_3”, que possuía prioridade 1, foi chamado e, por último, foi chamado o “Long_event_2”. Note que, em cada início e fim de evento, foi comparado o tempo real e o tempo do clock.

INICIO: Wed Nov 24 21:13:53 2021

CHAMADAS ESCALONADAS DA FUNÇÃO: Wed Nov 24 21:13:53 2021 0.390625

INICIO DO EVENTO: Wed Nov 24 21:13:55 2021 0.390625 LISTA DE INFORMAÇÕES DE ARQUIVOS E DIRETÓRIOS

Tamanho	Data de Modificação	Data de Criação	Nome
1.06 KB	Tue Nov 16 23:35:22 2021	Tue Nov 16 23:35:23 2021	aula 2.txt
1.06 KB	Tue Nov 16 23:35:31 2021	Tue Nov 16 23:35:32 2021	aula.txt
1.90 KB	Thu Sep 23 07:15:32 2021	Fri Nov 12 12:07:08 2021	Botão.py
37.35 KB	Thu Sep 23 07:21:42 2021	Thu Nov 18 08:46:44 2021	Capa TP_AT - 2021.docx
0.02 KB	Fri Sep 10 07:51:43 2021	Fri Nov 12 12:07:08 2021	conta.py
0.34 KB	Fri Aug 13 08:11:05 2021	Fri Nov 12 12:07:08 2021	CPU INFO.py
1.89 KB	Fri Sep 10 11:50:57 2021	Fri Nov 12 12:07:08 2021	CPU pygame.py
0.19 KB	Fri Jul 23 10:55:13 2021	Fri Nov 12 12:07:08 2021	CUMSUM.py
0.67 KB	Thu Nov 18 08:44:17 2021	Mon Nov 22 19:37:23 2021	Etapa 4 Ex 7.py
1.17 KB	Fri Nov 19 01:32:02 2021	Mon Nov 22 19:34:44 2021	Etapa 4 Ex 10 PB.py
0.40 KB	Fri Nov 12 11:26:42 2021	Wed Nov 24 02:25:21 2021	Etapa 4 Ex 3 PB.py
0.69 KB	Tue Nov 16 23:10:51 2021	Mon Nov 22 19:35:39 2021	Etapa 4 Ex 4 PB.py
0.36 KB	Thu Nov 18 20:08:37 2021	Mon Nov 22 19:34:44 2021	Etapa 4 Ex 8 PB.py
0.68 KB	Mon Nov 22 18:13:31 2021	Wed Nov 24 00:25:01 2021	Etapa 4 Ex 9 PB.py
0.03 KB	Wed Nov 24 05:28:04 2021	Wed Nov 24 05:28:14 2021	import time.py
0.08 KB	Thu Aug 26 12:11:06 2021	Fri Nov 12 12:07:08 2021	IP.py
3.36 KB	Tue Aug 17 09:49:15 2021	Fri Nov 12 12:07:08 2021	Jogo do quadradinho.py
239.70 KB	Mon Sep 20 07:11:12 2021	Fri Nov 12 12:07:08 2021	Marcelo_Oliveira_DR1_AT.pdf
0.65 KB	Thu Aug 26 11:33:49 2021	Fri Nov 12 12:07:08 2021	Memória - 1.py
0.00 KB	Fri Nov 12 12:08:31 2021	Fri Nov 12 12:09:06 2021	Novo(a) Documento do Microsoft Word.docx
0.61 KB	Fri Nov 12 11:40:30 2021	Fri Nov 12 12:44:09 2021	OS Etapa 4 Ex 2 PB.py
0.11 KB	Thu Sep 9 09:00:10 2021	Fri Nov 12 12:07:08 2021	pandas.py
1.05 KB	Thu Sep 9 19:51:40 2021	Fri Nov 12 12:07:08 2021	q14.py
1.42 KB	Thu Sep 9 20:02:51 2021	Fri Nov 12 12:07:08 2021	Questão 13.py
1.17 KB	Thu Sep 9 20:28:25 2021	Fri Nov 12 12:07:08 2021	Questão 14.py
0.38 KB	Fri Jul 23 10:40:23 2021	Fri Nov 12 12:07:08 2021	Rasc.py
0.04 KB	Fri Nov 12 11:36:37 2021	Fri Nov 12 12:07:08 2021	rasc2.py
1.90 KB	Fri Aug 13 09:02:55 2021	Fri Nov 12 12:07:08 2021	Simple gui.py

FIM DO EVENTO: Wed Nov 24 21:13:59 2021 0.4375

EVENTO: Wed Nov 24 21:14:01 2021 0.4375 LISTA DE INFORMAÇÕES DE UM PROCESSO

Nome: notepad.exe

Executável: C:\Windows\SysWOW64\notepad.exe

Tempo de criação: Wed Nov 24 21:14:01 2021

Tempo de usuário: 0.0 s

Tempo de sistema: 0.0 s

Percentual de uso de CPU:: 25.0 %

Percentual de uso de memória: 0.23 %

Uso de memória: 13.95 MB

Número de threads: 7

FIM DO EVENTO: Wed Nov 24 21:14:02 2021 0.46875

Arquivo	Editar	Formatar	Exibir	Ajuda					
INICIO DO EVENTO: Wed Nov 24 21:14:02 2021 0.46875 LISTA DE INFORMAÇÕES DE PROCESSOS									
PID	#	THREADS	DATA DE CRIAÇÃO	T. USU.	T. SIS.	MEM. (%)	RSS	VMS	EXECUTÁVEL
4	152	Wed Dec 31 21:00:00 1969	0.00	7906.52	0.09	%	5.10 MB	0.20 MB	
68	8	Wed Nov 10 00:25:27 2021	6.72	8.84	0.19	%	11.06 MB	7.42 MB	C:\Windows\System32\taskhostw.exe
100	4	Wed Nov 10 00:21:43 2021	0.00	11.73	0.54	%	32.48 MB	5.69 MB	Registry
312	23	Wed Nov 10 00:23:27 2021	198.02	105.22	1.00	%	59.66 MB	54.49 MB	C:\Program Files\Avast Software\Avast\aswEngSrv.exe
356	14	Wed Nov 17 20:53:11 2021	0.12	0.38	0.01	%	0.56 MB	18.39 MB	C:\Program Files\WindowsApps\
Microsoft.ZuneMusic_10.21102.11411.0_x64_8wekyb3d8bwe\Music.UI.exe									
396	18	Wed Nov 10 00:22:51 2021	101.45	166.25	0.46	%	27.39 MB	21.62 MB	C:\Windows\System32\svchost.exe
488	5	Wed Nov 10 00:22:52 2021	2.02	1.14	0.11	%	6.39 MB	6.07 MB	C:\Windows\System32\fontdrvhost.exe
520	2	Wed Nov 10 00:22:01 2021	0.00	0.23	0.01	%	0.36 MB	1.04 MB	C:\Windows\System32\smss.exe
560	5	Wed Nov 10 00:22:52 2021	0.91	1.08	0.01	%	0.74 MB	1.77 MB	C:\Windows\System32\fontdrvhost.exe
576	1	Wed Nov 10 00:26:50 2021	0.11	0.64	0.07	%	4.39 MB	2.26 MB	C:\Windows\System32\SecurityHealthSystray.exe
660	15	Wed Nov 10 00:22:49 2021	3.48	15.70	0.04	%	2.39 MB	2.26 MB	C:\Windows\System32\csrss.exe
752	1	Wed Nov 10 00:22:50 2021	0.03	0.09	0.03	%	1.68 MB	1.33 MB	C:\Windows\System32\wininit.exe
764	14	Wed Nov 10 00:22:50 2021	3.33	208.53	0.05	%	2.91 MB	2.62 MB	C:\Windows\System32\csrss.exe
804	11	Wed Nov 10 00:22:53 2021	244.70	162.95	0.29	%	17.04 MB	17.29 MB	C:\Windows\System32\svchost.exe
832	5	Wed Nov 10 00:22:54 2021	12.73	17.20	0.06	%	3.87 MB	2.57 MB	C:\Windows\System32\svchost.exe
836	7	Wed Nov 10 00:22:50 2021	41.27	81.70	0.12	%	7.41 MB	6.50 MB	C:\Windows\System32\services.exe
844	7	Wed Nov 10 00:22:50 2021	66.27	53.50	0.26	%	15.59 MB	10.09 MB	C:\Windows\System32\lsass.exe
932	3	Wed Nov 10 00:22:51 2021	0.47	2.19	0.09	%	5.29 MB	2.58 MB	C:\Windows\System32\winlogon.exe
948	2	Wed Nov 10 00:25:52 2021	21.59	57.83	0.11	%	6.32 MB	3.04 MB	C:\Windows\System32\svchost.exe
1044	10	Mon Nov 22 16:44:38 2021	13.44	73.89	0.18	%	10.95 MB	5.37 MB	C:\Windows\System32\svchost.exe
1076	14	Wed Nov 10 00:22:54 2021	2718.16	1582.19	0.56	%	33.12 MB	60.09 MB	C:\Windows\System32\dmw.exe
1108	7	Wed Nov 10 00:22:57 2021	28.52	29.31	0.07	%	4.41 MB	3.50 MB	C:\Windows\System32\svchost.exe
1132	4	Wed Nov 10 00:22:55 2021	1.88	1.33	0.09	%	5.41 MB	2.55 MB	C:\Windows\System32\svchost.exe
1172	16	Wed Nov 10 00:22:55 2021	0.09	0.09	0.08	%	4.55 MB	3.00 MB	C:\Windows\System32\svchost.exe
1340	3	Wed Nov 10 00:22:55 2021	1.80	1.19	0.08	%	5.02 MB	2.34 MB	C:\Windows\System32\svchost.exe
1348	1	Wed Nov 10 00:47:07 2021	0.28	0.48	0.10	%	5.92 MB	1.78 MB	C:\Windows\System32\RuntimeBroker.exe
1372	9	Wed Nov 10 00:22:55 2021	17.06	30.56	0.19	%	11.20 MB	14.99 MB	C:\Windows\System32\svchost.exe
1420	9	Mon Nov 22 16:14:06 2021	1.23	2.03	0.15	%	9.21 MB	12.54 MB	C:\Users\win10\AppData\Roaming\Zoom\data\cef_x86\zcefAgent.exe
1540	2	Wed Nov 10 00:22:55 2021	0.25	0.92	0.09	%	5.32 MB	3.32 MB	C:\Windows\System32\svchost.exe
1548	3	Wed Nov 10 00:22:55 2021	0.19	0.88	0.03	%	1.79 MB	1.26 MB	C:\Windows\System32\svchost.exe
1556	3	Wed Nov 10 00:22:55 2021	1212.92	746.05	0.55	%	32.56 MB	29.23 MB	C:\Windows\System32\svchost.exe
1564	10	Wed Nov 10 00:22:55 2021	0.75	1.02	0.14	%	8.41 MB	4.88 MB	C:\Program Files\Avast Software\Avast\wsc_proxy.exe

FIM DO EVENTO: Wed Nov 24 21:14:06 2021 4.078125

Tempo real x clock da CPU

Quando as funções do “sched” foram executadas, foi apresentado, no início e no fim do evento, o tempo real e o clock da CPU. Existe uma diferença nesses dois tempos. O tempo real calcula o tempo de início e fim do evento. Já o clock calcula o tempo de processamento do evento. Isso pode ser notado quando utilizamos a função “time.sleep()”, pois quando utilizamos essa função, não há processamento, logo não há variação do clock da CPU.

Redes e Subredes

Nessa fase, foi criado funções para explorar as redes e subredes que se encontram as máquinas. Os módulos utilizados foram “nmap”, “subprocess”, “platform” e “os”.

Na primeira estrutura de código foi criado três funções para retornar informações sobre as máquinas que estão na subrede.

A primeira função é o “código_retorna_ping”, função essa que vai fazer a chamada do ping. Após, foi criado a função “verifica_hosts”, que vai varrer a subrede para buscar outros IPs que estão na subrede e retorna uma lista com todos os IPs ativos na subrede. Por fim, a função “obter_hostnames”, função que retorna o nome das máquinas associadas aos IPs encontrados.

```
def retorna_codigo_ping(hostname):
    """Usa o utilitario ping do sistema operacional para encontrar

    plataforma = platform.system()
    args = []
    if plataforma == "Windows":
        args = ["ping", "-n", "1", "-l", "1", "-w", "100", hostname]

    else:
        args = ['ping', '-c', '1', '-W', '1', hostname]

    ret_cod = subprocess.call(args,
                               stdout=open(os.devnull, 'w'),
                               stderr=open(os.devnull, 'w'))

    return ret_cod

def verifica_hosts(base_ip):
    """Verifica todos os host com a base_ip entre 1 e 255 retorna uma lista com todos os host que ti
    print("Mapeando\r")
    host_validos = []
    return_codes = dict()
    for i in range(1, 255):

        return_codes[base_ip + '{0}'.format(i)] = retorna_codigo_ping(base_ip + '{0}'.format(i))
        if i % 20 == 0:
            print(".", end = "")
            if return_codes[base_ip + '{0}'.format(i)] == 0:
                host_validos.append(base_ip + '{0}'.format(i))
    print("\nMapping ready...")

    return host_validos
```

```
def obter_hostnames(host_validos):
    nm = nmap.PortScanner()
    for i in host_validos:
        try:
            nm.scan(i)
            print('O IP possui o nome ', nm[i].hostname())
            nm.scan(i)
            #print('O IP', host, 'possui o nome', nm[i].hostname())
        except:
            print('O IP deu problema.')
            pass
```

Após a criação da função foram feitas as chamadas.

```
# Chamadas
ip_string = input("Entre com o ip alvo: ")
ip_lista = ip_string.split('.')
base_ip = ".".join(ip_lista[0:3]) + '.'
print("O teste será feito na sub rede: ", base_ip)
host_validos = verifica_hosts(base_ip)
print ("Os host válidos são: ", host_validos)

print('Iniciando nmap.PortScanner')
obter_hostnames(host_validos)
print()
print('FIM.')
```

O resultado ficou assim:

```
>>> %Run 'TP6 EC1.py'

Entre com o ip alvo: 192.168.0.59
O teste será feito na sub rede: 192.168.0.
Mapeando
.....
Mapping ready...
Os host válidos são: ['192.168.0.1', '192.168.0.28', '192.168.0.31', '192.168.0.59']
Iniciando nmap.PortScanner
O IP possui o nome cable.box
O IP possui o nome
O IP possui o nome
O IP possui o nome DESKTOP-PEOF25U.box

FIM.
```

Outra estrutura de código criada foi para apresentar informações sobre as portas dos diferentes IPs obtidos na sub rede. A estrutura de código é praticamente igual a outra, porém na função “obter_hostnames” foi adicionado alguns comandos para informar sobre as portas dos IPs.

```
def obter_hostnames(host_validos):
    nm = nmap.PortScanner()
    for i in host_validos:
        try:
            nm.scan(i)
            print('O IP possui o nome ', nm[i].hostname())
            print(nm[i].hostname())
            for proto in nm[i].all_protocols():
                print('-----')
                print('Protocolo : %s' % proto)

                lport = nm[i][proto].keys()
                #lport.sort()
                for port in lport:
                    print ('Porta: %s\t Estado: %s' % (port, nm[i][proto][port]['state']))
            nm.scan(i)
            #print('O IP', host, 'possui o nome', nm[i].hostname())
        except:
            print('O IP deu problema.')
            pass
```

O resultado fica assim:

```
.....
Entre com o ip alvo: 192.168.0.59
O teste será feito na sub rede: 192.168.0.
Mapeando
.....
Mapping ready...
Os host válidos são: ['192.168.0.1', '192.168.0.28', '192.168.0.31', '192.168.0.59']
Iniciando nmap.PortScanner
O IP possui o nome cable.box
cable.box
-----
Protocolo : tcp
Porta: 22 Estado: filtered
Porta: 23 Estado: filtered
Porta: 53 Estado: open
Porta: 80 Estado: open
Porta: 111 Estado: filtered
Porta: 443 Estado: open
Porta: 5000 Estado: open
O IP possui o nome

-----
Protocolo : tcp
Porta: 62078 Estado: open
O IP possui o nome

-----
Protocolo : tcp
Porta: 5555 Estado: open
O IP possui o nome DESKTOP-PEOF25U.box
DESKTOP-PEOF25U.box
-----
Protocolo : tcp
Porta: 135 Estado: open
Porta: 139 Estado: open
Porta: 445 Estado: open
Porta: 1688 Estado: open
```

Nmap x Ping

Como foi mencionado anteriormente, nessa fase, foi utilizado a biblioteca “nmap” e a ferramenta ping. As duas são utilizadas para verificar os IPs ativos na subrede. Porém, a “nmap” possui outras informações além da verificação de um IP, como o nome da máquina para qual o IP está associado, a quantidade de portas testadas e quais portas estão abertas. Logo, o “nmap” traz informações mais completas que o “ping”.

Informações de Rede

Nessa fase, foram feitos 3 estruturas de códigos, cujo objetivo era buscar Informações sobre a rede. O módulo “psutil” foi utilizado para buscar as informações de rede. A primeira função do “psutil” utilizada foi “psutil.net_if_addrs()”. Essa função retorna as seguintes informações:

- **family:** a família dos endereços de IP. Pode ser AF_INET (camada de rede - IPV4), AF_INET6 (camada de rede - IPV6) ou psutil.AF_LINK (camada de enlace - MAC).
- **address:** o endereço de IP.
- **netmask:** máscara do endereço (pode ser *None*).
- **broadcast:** endereço de *broadcast* (pode ser *None*).
- **ptp:** endereço para comunicação ponto a ponto (*peer-to-peer*). Pode ser *None*.

A estrutura de código ficou da seguinte forma:

```
import psutil
import time
interfaces = psutil.net_if_addrs()
nomes = []
# Obtém os nomes das interfaces primeiro
for i in interfaces:
    nomes.append(str(i))
# Depois, imprimir os valores:
for i in nomes:
    print(i+":")
    for j in interfaces[i]:
        print("\t"+str(j))
    print('-----')
```

Essa estrutura retorna as seguintes informações:

```

Ethernet:
    snicaddr(family=<AddressFamily.AF_LINK: -1>, address='20-89-84-0C-AF-29', netmask=None, broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET: 2>, address='169.254.16.146', netmask='255.255.0.0', broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET6: 23>, address='fe80::c50:905d:1370:1092', netmask=None, broadcast=None, ptp=None)
-----
Conexão Local* 1:
    snicaddr(family=<AddressFamily.AF_LINK: -1>, address='1E-C9-D3-29-A3-99', netmask=None, broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET: 2>, address='169.254.3.127', netmask='255.255.0.0', broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET6: 23>, address='fe80::f938:e201:7ab4:37f', netmask=None, broadcast=None, ptp=None)
-----
Conexão Local* 2:
    snicaddr(family=<AddressFamily.AF_LINK: -1>, address='2E-C9-D3-29-A3-99', netmask=None, broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET: 2>, address='169.254.172.133', netmask='255.255.0.0', broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET6: 23>, address='fe80::a44b:110a:7443:ac85', netmask=None, broadcast=None, ptp=None)
-----
Wi-Fi:
    snicaddr(family=<AddressFamily.AF_LINK: -1>, address='5C-C9-D3-29-A3-99', netmask=None, broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET: 2>, address='192.168.0.59', netmask='255.255.255.0', broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET6: 23>, address='2804:14d:5c31:9b55:9123', netmask=None, broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET6: 23>, address='2804:14d:5c31:9b55:2118:cac0:b4da:2a45', netmask=None, broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET6: 23>, address='2804:14d:5c31:9b55:4473:3d55:baaf:b65b', netmask=None, broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET6: 23>, address='fe80::2118:cac0:b4da:2a45', netmask=None, broadcast=None, ptp=None)
-----
Loopback Pseudo-Interface 1:
    snicaddr(family=<AddressFamily.AF_INET: 2>, address='127.0.0.1', netmask='255.0.0.0', broadcast=None, ptp=None)
    snicaddr(family=<AddressFamily.AF_INET6: 23>, address='::1', netmask=None, broadcast=None, ptp=None)

```

A segunda função utilizada do modulo “psutil” foi a “psutil.net_io_counters()”. Essa função retorna estatísticas de entrada e saída de dados através da interface de rede principal. Essas estatísticas são:

- **bytes_sent**: número de bytes enviados.
- **bytes_recv**: número de bytes recebidos.
- **packets_sent**: número de pacotes enviados.
- **packets_recv**: número de pacotes recebidos.
- **errin**: número total de erros durante o recebimento.
- **errout**: número total de erros durante o envio.
- **dropin**: número total de pacotes de entrada (recebimento) que foram descartados.
- **dropout**: número total de pacotes de saída (envio) que foram descartados.

A estrutura de código criado em cima dessa função retorna as informações estatísticas de entrada e saída de dados em cada uma das interfaces de rede da máquina a cada segundo por 3 vezes. Ela fica da seguinte forma:

```

import psutil
import time

io_status = psutil.net_io_counters(pernic=True)
nomes = []
for i in io_status:
    nomes.append(str(i))
for j in nomes:
    print(j)
    print("\t"+str(io_status[j]))
for i in range(4):
    time.sleep(1)
    io_status = psutil.net_io_counters(pernic=True)
    for j in nomes:
        print(j)
        print("\t"+str(io_status[j]))

```

Resultado fica da seguinte forma:

```

Ethernet
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Conexão Local* 1
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Conexão Local* 2
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Wi-Fi
  snetio(bytes_sent=133009668, bytes_recv=3077740188, packets_sent=1088827, packets_recv=2760581, errin=0, errout=0, dropin=0, dropout=0)
Loopback Pseudo-Interface 1
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
-----
Ethernet
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Conexão Local* 1
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Conexão Local* 2
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Wi-Fi
  snetio(bytes_sent=133010211, bytes_recv=3077740655, packets_sent=1088831, packets_recv=2760584, errin=0, errout=0, dropin=0, dropout=0)
Loopback Pseudo-Interface 1
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
-----
Ethernet
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Conexão Local* 1
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Conexão Local* 2
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)
Wi-Fi
  snetio(bytes_sent=133010211, bytes_recv=3077740655, packets_sent=1088831, packets_recv=2760584, errin=0, errout=0, dropin=0, dropout=0)
Loopback Pseudo-Interface 1
  snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)

```

A última estrutura de código foi criada pra buscar informações de processos usando redes. Essas informações são:

- **Tipo do endereço (End.):** pode ser IPv4 ou IPv6 (ou Unix, para sistemas Unix).
- **Tipo de conexão (Tipo):** pode ser TCP, UDP ou IP.
- **Status da conexão (Status):** existem vários valores possíveis, como, por exemplo, ESTABLISHED, CLOSE, LISTEN, NONE.
- **Endereço local:** valor do endereço IP (v4 ou v6) local.

- **Porta local:** valor do número de porta da conexão local.
- **Endereço remoto:** valor do endereço IP (v4 ou v6) remoto.
- **Porta remota:** valor do número de porta da conexão remoto

Nessa estrutura de código, foi utilizado, além do módulo “psutil”, o módulo “socket”. Do “psutil”, foi utilizado a função “psutil.pids()” para obter os PIDs dos processos dos sistemas. Além dele, foi utilizado o “psutil.Process()”. Do módulo “socket”, duas funções foram criadas: “obtem_nome_familia()” e “obtem_tipo_socket()”.

A função “obtem_nome_familia()” foi criada pra retornar o tipo de endereço.

```
import psutil
import socket

def obtem_nome_familia(familia):
    if familia == socket.AF_INET:
        return("IPv4")
    elif familia == socket.AF_INET6:
        return("IPv6")
    elif familia == socket.AF_UNIX:
        return("Unix")
    else:
        return("-")
```

Já a função “obtem_tipo_socket()” foi criada pra retornar o tipo de conexão.

```
def obtem_tipo_socket(tipo):
    if tipo == socket.SOCK_STREAM:
        return("TCP")
    elif tipo == socket.SOCK_DGRAM:
        return("UDP")
    elif tipo == socket.SOCK_RAW:
        return("IP")
    else:
        return("-")
```

```

for i in psutil.pids():
    p = psutil.Process(i)
    conn = p.connections()
    if len(conn)>0:
        if conn[0].status.ljust(13) != 'ESTABLISHED ':
            endl = conn[0].laddr.ip.ljust(11)
            portl = str(conn[0].laddr.port).ljust(5)
            endr = conn[0].laddr.ip.ljust(13)
            portr = str(conn[0].laddr.port).ljust(5)
            print(str(i).ljust(5), ' End. tipo status', ' Endereço Local', ' Porta L.', ' Endereço Remoto', ' Porta R. ')
            print(' ', obtem_nome_familia(conn[0].family), ' ' + obtem_tipo_socket(conn[0].type),
                  ' ' + conn[0].status.ljust(13), endl, ' ' + portl, ' ' + endr, ' ' + portr)

```

O resultado fica da seguinte forma:

0	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv4	TCP	TIME_WAIT	192.168.0.59	1045	192.168.0.59	1045
4	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv4	UDP	NONE	192.168.0.59	137	192.168.0.59	137
752	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv4	TCP	LISTEN	0.0.0.0	49665	0.0.0.0	49665
804	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv4	TCP	LISTEN	0.0.0.0	135	0.0.0.0	135
836	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv6	TCP	LISTEN	::	49704	::	49704
844	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv6	TCP	LISTEN	::	49664	::	49664
1372	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv6	TCP	LISTEN	::	49667	::	49667
1596	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv4	UDP	NONE	0.0.0.0	5353	0.0.0.0	5353
1764	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv4	UDP	NONE	0.0.0.0	57862	0.0.0.0	57862
1792	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv6	TCP	LISTEN	::	49666	::	49666
2080	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv6	UDP	NONE	:::1	1900	:::1	1900
2136	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv6	UDP	NONE	::	3702	::	3702
2272	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv6	TCP	ESTABLISHED	::	3702	::	3702
2304	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.
	IPv6	TCP	LISTEN	::	1688	::	1688
2592	End.	tipo	status	Endereço Local	Porta L.	Endereço Remoto	Porta R.

Serviços de Rede

Nessa fase, 2 estruturas de código foram montadas para fazer o transporte de informações na rede através do serviço cliente-servidor. Para isso, foi utilizado o módulo “socket”. Além desse módulo, foi utilizado também outros como “psutil”, “platform”, “os”, “time”, “sys” e “subprocess” para buscar as informações que foram solicitadas.

A primeira estrutura de código montada foi a do modelo cliente. Para isso, utilizamos as “socket.socket()” para criar o socket.

A estrutura de código para estabelecer conexão ficou da seguinte forma:

```
# Cliente
import socket, sys
# Cria o socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    # Tenta se conectar ao servidor
    s.connect(("192.168.0.59", 9999))

except Exception as erro:
    print(str(erro))
    sys.exit(1) # Termina o programa
```

No modelo servidor, a estrutura de código para estabelecer a conexão da seguinte forma:

```
# Servidor
import socket, psutil, platform, os, time, subprocess

# Cria o socket
socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Obtem o nome da máquina
host = socket.gethostname()
porta = 9999
# Associa a porta
socket_servidor.bind((host, porta))
# Escutando...
socket_servidor.listen()
print("Servidor de nome", host, "esperando conexão na porta", porta)
# Aceita alguma conexão
(socket_cliente, addr) = socket_servidor.accept()
print("Conectado a:", str(addr))
```

No modelo servidor, foi criado o socket, foi obtido o nome da máquina e a porta de acesso.

As informações que foram transportadas nessa etapa foram informações de arquivos e diretórios, como nome, data de criação, data de modificação e tamanho. Além dos arquivos e diretórios, também houve transporte de informações dos processos em execução, como PID, nome do executável, tempo de CPU de usuário, tempo de CPU de sistema, percentual de uso de CPU, uso de memória, percentual do uso de memória principal e número de threads.

```
opcao = input('O que você deseja? ')
if 'lista de informações de arquivos e diretórios' == opcao:
    print("Para encerrar, digite '$'")
    msg = input()
    # Envia mensagem codificada em bytes ao servidor
    s.send(msg.encode('utf-8'))
    while msg != '$':
        msg = s.recv(100000)
        print(msg.decode('utf-8')) #.decode('utf-8')
        print("Para encerrar, digite '$'")
        msg = input()
    # Envia mensagem codificada em bytes ao servidor
    s.send(msg.encode('utf-8'))
    # Fecha conexão com o servidor
    s.close()
elif 'lista de informação de processo' == opcao:
    print("Para encerrar, digite '$'")
    msg = input()
    # Envia mensagem codificada em bytes ao servidor
    s.send(msg.encode('utf-8'))
    while msg != '$':
        msg = s.recv(100000)
        print(msg.decode('utf-8')) #.decode('utf-8')
        print("Para encerrar, digite '$'")
```

```

while True:
    msg = socket_cliente.recv(100000)
    # Decodifica mensagem em UTF-8:
    if '$' == msg.decode('utf-8'): #Termino do cliente
        print("Fechando conexao com", str(addr), "...")
        socket_cliente.close()
        break
    nome = msg.decode('utf-8')
    print(nome)
    lista = os.listdir()
    dic = {}
    try:
        for i in lista:
            if nome == i: #Termino do cliente
                if os.path.isfile(nome):
                    dic[nome] = []
                    dic[nome].append(os.stat(nome).st_size) # Tamanho
                    dic[nome].append(os.stat(nome).st_atime) # Tempo de criação
                    dic[nome].append(os.stat(nome).st_mtime) # Tempo de modificação
                kb = str(round(dic[nome][0]/1024, 2)) + ' KB'
                informacao = 'nome: ' + nome + '\n' + 'Tamanho: ' + kb + '\n'
                informacao = informacao + 'Data de criação: ' + str(time.ctime(dic[nome][1])) + '\n'
                informacao = informacao + 'Data de modificação: ' + str(time.ctime(dic[nome][2])) + '\n'
                socket_cliente.send(informacao.encode('utf-8')) # Envia resposta
                print(informacao)
    except:
        pass

try:
    pid = subprocess.Popen(nome).pid
    p = psutil.Process(pid)
    perc_mem = '{:.2f}'.format(p.memory_percent())
    mem = '{:.2f}'.format(p.memory_info().rss/1024/1024)

    informacao_p = 'nome: ' + p.name() + '\n' + 'Executável: ' + p.exe() + '\n'
    informacao_p = informacao_p + '\n' + 'Tempo de criação: ' + time.ctime(p.create_time())
    informacao_p = informacao_p + '\n' + 'Tempo de CPU de sistema: ' + str(p.cpu_times().system) + 's'
    informacao_p = informacao_p + '\n' + 'Tempo de CPU de usuário: ' + str(p.cpu_times().user) + 's'
    informacao_p = informacao_p + '\n' + 'Percentual de uso de CPU: ' + str(p.cpu_percent(interval = 1.0)) + '%'
    informacao_p = informacao_p + '\n' + 'Percentual de uso de memória: ' + str(perc_mem) + '%'
    informacao_p = informacao_p + '\n' + 'Uso de memória: ' + str(mem) + 'MB'
    informacao_p = informacao_p + '\n' + 'Número de threads: ' + str(p.num_threads())
    socket_cliente.send(informacao_p.encode('utf-8'))
except:
    pass

```

O resultado fica da seguinte forma:

Cliente

```

>>> %Run Client.py

O que você deseja? lista de informações de arquivos e diretórios
Para encerrar, digite '$'
Formulário.html
nome: Formulário.html
Tamanho: 2.7 KB
Data de criação: Tue Nov 30 04:32:06 2021
Data de modificação: Tue Aug 31 16:14:56 2021

```

Servidor

```
Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\Users\win10\.vscode\HTML> & 'C:\Users\win10\AppData\Local\Microsoft\WindowsApps\python3.9.exe' 'c:\Users\win10\.vscode\extensions\ms-python
.python-2021.11.1422169775\pythonFiles\lib\python\debugpy\launcher' '1118' '--' 'c:\Users\win10\Thonny\# Servidor.py'
Servidor de nome DESKTOP-PEOF25U esperando conexão na porta 9999
Conectado a: ('192.168.0.59', 1124)
Formulário.html
nome: Formulário.html
Tamanho: 2.7 KB
Data de criação: Tue Nov 30 04:32:06 2021
Data de modificação: Tue Aug 31 16:14:56 2021
```

Fase final:

Na fase final do projeto, compilado todas as estruturas de código criadas nas fases anteriores dentro da estrutura de código do servidor. Além disso foi criado também as chamadas das estruturas de código do servidor no código do cliente. Na estrutura de código do cliente, também foi inserido um menu, para facilitar as chamadas das estruturas de código no servidor.

O código do cliente ficou da seguinte forma:

```
# Cliente
import socket, sys, pickle
# Cria o socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    # Tenta se conectar ao servidor
    s.connect(("192.168.0.129", 9999))
except Exception as erro:
    print(str(erro))
    sys.exit(1) # Termina o programa
msg = input("Menu: \n [1] Informações de arquivos e diretórios \n [2] Informações de processo \n [3] Escalonamento de Chamadas\n")
# Envia mensagem codificada em bytes ao servidor
s.send(msg.encode('utf-8'))
while msg != '9':
    if msg == '1':
        bytes = s.recv(10000)
        lista_arq = pickle.loads(bytes)
        for i in range(len(lista_arq)):
            print(lista_arq[i])
        msg = input("Menu: \n [1] Informações de arquivos e diretórios \n [2] Informações de processo \n [3] Escalonamento de Chamadas\n")
        # Envia mensagem codificada em bytes ao servidor
        s.send(msg.encode('utf-8'))
    elif msg == '2':
        bytes = s.recv(100000)
        lista_proc = pickle.loads(bytes)
        for i in range(len(lista_proc)):
            print(lista_proc[i])
        msg = input("Menu: \n [1] Informações de arquivos e diretórios \n [2] Informações de processo \n [3] Escalonamento de Chamadas\n")
        # Envia mensagem codificada em bytes ao servidor
        s.send(msg.encode('utf-8'))
    elif msg == '3':
        bytes = s.recv(100000)
        lista_esc = pickle.loads(bytes)
        for i in range(len(lista_esc)):
            print(lista_esc[i])
        msg = input("Menu: \n [1] Informações de arquivos e diretórios \n [2] Informações de processo \n [3] Escalonamento de Chamadas\n")
        # Envia mensagem codificada em bytes ao servidor
        s.send(msg.encode('utf-8'))
    elif msg == '4':
        bytes = s.recv(100000)
        if 'Ok... Entre com um IP alvo(o processo pode levar alguns minutos): ' == bytes.decode('utf-8'):
            print(bytes)
            msg = input()
            s.send(msg.encode('utf-8'))
            bytes = s.recv(100000)
            lista_proc = pickle.loads(bytes)
            for i in range(len(lista_proc)):
                print(lista_proc[i])
            msg = input("Menu: \n [1] Informações de arquivos e diretórios \n [2] Informações de processo \n [3] Escalonamento de Chamadas\n")
            # Envia mensagem codificada em bytes ao servidor
            s.send(msg.encode('utf-8'))
    elif msg == '5':
        bytes = s.recv(100000)
        lista_ir = pickle.loads(bytes)
        for i in range(len(lista_ir)):
            print(lista_ir[i])
        msg = input("Menu: \n [1] Informações de arquivos e diretórios \n [2] Informações de processo \n [3] Escalonamento de Chamadas\n")
        # Envia mensagem codificada em bytes ao servidor
        s.send(msg.encode('utf-8'))
# Fecha conexão com o servidor
s.close()
```

O código do servidor ficou da seguinte forma:

```
# Servidor
import socket, psutil, platform, os, time, subprocess, sched, nmap, pickle

# Cria o socket
socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Obtem o nome da máquina
host = socket.gethostname()
porta = 9999
# Associa a porta
socket_servidor.bind((host, porta))
# Escutando...
socket_servidor.listen()
print("Servidor de nome", host, "esperando conexão na porta", porta)
# Aceita alguma conexão
(socket_cliente, addr) = socket_servidor.accept()
print("Conectado a:", str(addr))

while True:
    msg = socket_cliente.recv(100000)
    # Decodifica mensagem em UTF-8:
    if '9' == msg.decode('utf-8'): #Termino do cliente
        print("Fechando conexao com", str(addr), "...")
        socket_cliente.close()
        break

    elif '1' == msg.decode('utf-8'):
        try:
            lista = os.listdir()
            dic = {} # cria dicionário
            lista_arq = []
            for i in lista: # Varia na lista dos arquivos e diretórios
                if os.path.isfile(i): # checa se é um arquivo
                    # Cria uma lista para cada arquivo. Esta lista contém o
                    # tamanho, data de criação e data de modificação.
                    dic[i] = []
                    dic[i].append(os.stat(i).st_size) # Tamanho
                    dic[i].append(os.stat(i).st_atime) # Tempo de criação
                    dic[i].append(os.stat(i).st_mtime) # Tempo de modificação

            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"
            lista_arq.append(titulo)
            for i in dic:
                kb = dic[i][0]/1024
                tamanho = '{:10}'.format(str('{:.2f}'.format(kb)+' KB'))
                arquivo = tamanho + str(time.ctime(dic[i][2])) + " " + str(time.ctime(dic[i][1])) + " " + i
                lista_arq.append(arquivo)
            bytes = pickle.dumps(lista_arq)
            socket_cliente.send(bytes) # Envia resposta

        except:
            lista = []
            lista.append('erro no servidor')
            print(lista)
            bytes = pickle.dumps(lista)
            socket_cliente.send(bytes)
```

```

elif '2' == msg.decode('utf-8'):
    try:
        lista = psutil.pids()
        lista_proc = []
        titulo = '{:7}'.format("PID")
        titulo = titulo + '{:11}'.format("# THREADS")
        titulo = titulo + '{:26}'.format("CRIAÇÃO")
        titulo = titulo + '{:9}'.format("T. USU.")
        titulo = titulo + '{:9}'.format("T. SIS.")
        titulo = titulo + '{:12}'.format("MEM. (%)")
        titulo = titulo + '{:12}'.format("RSS")
        titulo = titulo + '{:12}'.format("VMS")
        titulo = titulo + " EXECUTÁVEL"
        lista_proc.append(titulo)
        def mostra_info(pid):
            try:
                p = psutil.Process(pid)
                texto = '{:6}'.format(pid)
                texto = texto + '{:11}'.format(p.num_threads())
                texto = texto + " " + time.ctime(p.create_time()) + " "
                texto = texto + '{:8.2f}'.format(p.cpu_times().user)
                texto = texto + '{:8.2f}'.format(p.cpu_times().system)
                texto = texto + '{:10.2f}'.format(p.memory_percent()) + " %"
                rss = p.memory_info().rss/1024/1024
                texto = texto + '{:10.2f}'.format(rss) + " MB"
                vms = p.memory_info().vms/1024/1024
                texto = texto + '{:10.2f}'.format(vms) + " MB"
                texto = texto + " " + p.exe()
                lista_proc.append(texto)
            except:
                pass
        for i in lista:

```

```

        for i in lista:
            mostra_info(i)
        bytes = pickle.dumps(lista_proc)
        socket_cliente.send(bytes) # Envia resposta
    except:
        lista = []
        lista.append('erro no servidor')
        bytes = pickle.dumps(lista)
        socket_cliente.send(bytes)

```

```

elif '3' == msg.decode('utf-8'):
    try:
        scheduler = sched.scheduler(time.time, time.sleep)
        lista_by = []

        def Long_event_1(name):
            resposta = 'INICIO DO EVENTO:' + str(time.ctime()) + str(time.process_time())
            print(resposta)
            lista_dir = []
            p_dir = ''
            nome = input()
            if os.path.isdir(nome):
                lista_dir.append(nome)
                somador = 0
                while lista_dir:
                    diretorio = lista_dir[0]
                    p_dir = os.path.join(p_dir, diretorio)
                    lista = os.listdir(p_dir)
                    for i in lista:
                        p = os.path.join(p_dir, i)
                        if os.path.isdir(p):
                            lista_dir.append(i)
                        elif os.path.isfile(p):
                            somador = somador + os.stat(p).st_size
                    lista_dir.remove(diretorio)
                print(nome, '\n', 'Tamanho:', str(round(somador/1024, 2)), ' KB')
                resposta = resposta + nome + ' ' + 'Tamanho:' + str(round(somador/1024, 2)) + ' KB' + ' '
            else:
                print("O diretório", '\'+ nome + '\', 'não existe.')
                resposta = resposta + "O diretório" + '\'+ nome + '\', 'não existe.'
            print('FIM DO EVENTO:', time.ctime(), time.process_time())
            print()

```

```

print()
resposta = resposta + 'FIM DO EVENTO:' + str(time.ctime()), str(time.process_time())
lista_by.append(resposta)

def Long_event_2(name):
    print ('EVENTO:', time.ctime(), time.process_time())
    pid = subprocess.Popen('notepad.exe').pid

    p = psutil.Process(pid)
    resposta_2 = 'Nome: ' + p.name() + '\n'
    resposta_2 = resposta_2 + 'Executável: ' + p.exe() + '\n'
    resposta_2 = resposta_2 + 'Data de criação: ' + str(time.ctime(p.create_time())) + '\n'
    resposta_2 = resposta_2 + 'Tempo de usuário: ' + str(p.cpu_times().user) + 's \n'
    resposta_2 = resposta_2 + 'Tempo de sistema: ' + str(p.cpu_times().system) + 's \n'
    resposta_2 = resposta_2 + 'Percentual de uso de CPU:: ' + str(p.cpu_percent(interval = 1.0)) + '
    perc_mem = '{:.2f}'.format(p.memory_percent())
    resposta_2 = resposta_2 + 'Percentual de uso de memória: ' + perc_mem + '% \n'
    mem = '{:.2f}'.format(p.memory_info().rss/1024/1024)
    resposta_2 = resposta_2 + 'Uso de memória: ' + mem + 'MB \n'
    resposta_2 = resposta_2 + 'Número de threads: ' + str(p.num_threads()) + '\n'
    resposta_2 = resposta_2 + 'FIM DO EVENTO:' + str(time.ctime()) + ' clock: ' + str(time.process_t
    print(resposta_2)
    lista_by.append(resposta_2)
    print()

```

```

inicio = 'INICIO:' + str(time.ctime())
lista_by.append(inicio)
print(inicio)
scheduler.enter(2, 3, Long_event_1, ('TAMANHO DE DIRETÓRIOS',))
scheduler.enter(8, 1, Long_event_2, ('LISTA DE INFORMAÇÕES DE UM PROCESSO',))
chamada = 'CHAMADAS ESCALONADAS DA FUNÇÃO:' + str(time.ctime()) + str(time.process_time())
print(chamada)
scheduler.run()
lista_by.append(chamada)
bytes = pickle.dumps(lista_by)
socket_cliente.send(bytes)

```

```

elif '4' == msg.decode('utf-8'):
    msg = 'Ok... Entre com um IP alvo(o processo pode levar alguns minutos): '
    socket_cliente.send(msg.encode('utf-8'))
    msg = socket_cliente.recv(100000)
    def obter_hostnames(host_validos):
        resposta = "O teste foi feito na sub rede: " + base_ip + '\n'
        resposta = resposta + "Os hosts válidos são: " + str(host_validos) + '\n'
        nm = nmap.PortScanner()
        for i in host_validos:
            try:
                nm.scan(i)
                resposta = resposta + '\n' + 'O IP possui o nome ' + str(nm[i].hostname()) + '\n'

                for proto in nm[i].all_protocols():
                    resposta = resposta + '-----' + '\n'
                    resposta = resposta + 'Protocolo : ' + proto + '\n'

                    lport = nm[i][proto].keys()

                    for port in lport:
                        resposta = resposta + 'Porta: ' + str(port) + ' ' + 'Estado: ' + str(nm[i][proto][port]) + '\n'
            except:
                resposta = resposta + 'O IP deu problema.'
                pass

        lis.append(resposta)

```



```
def retorna_codigo_ping(hostname):
    """Usa o utilitario ping do sistema operacional para encontrar o host. ('-c 5') indica, em sistema

    plataforma = platform.system()
    args = []
    if plataforma == "Windows":
        args = ["ping", "-n", "1", "-l", "1", "-w", "100", hostname]

    else:
        args = ['ping', '-c', '1', '-W', '1', hostname]

    ret_cod = subprocess.call(args,
                               stdout=open(os.devnull, 'w'),
                               stderr=open(os.devnull, 'w'))

    return ret_cod
```

```
def verifica_hosts(base_ip):
    """Verifica todos os host com a base_ip entre 1 e 255 retorna uma lista com todos os host que tivera
    print("Mapeando\n")
    host_validos = []
    return_codes = dict()
    for i in range(1, 255):

        return_codes[base_ip + '{0}'.format(i)] = retorna_codigo_ping(base_ip + '{0}'.format(i))
        if i % 20 == 0:
            print(".", end = "")
            if return_codes[base_ip + '{0}'.format(i)] == 0:
                host_validos.append(base_ip + '{0}'.format(i))
    print("\nMapping ready...")

    return host_validos

# Chamadas
lis = []
ip_string = msg.decode('utf-8')
ip_lista = ip_string.split('.')
base_ip = ".".join(ip_lista[0:3]) + '.'
print("O teste será feito na sub rede: ", base_ip)
host_validos = verifica_hosts(base_ip)
print ("Os host válidos são: ", host_validos)

print('Iniciando nmap.PortScanner')
obter_hostnames(host_validos)
print()
print('FIM.')
bytes = pickle.dumps(lis)
socket_cliente.send(bytes)
```

ivo Atual (Thonny) In 13, Col 37 Espacos: 4 UTF-8 CRI

```
elif '5' == msg.decode('utf-8'):
    interfaces = psutil.net_if_addrs()
    nomes = []
    lista = []
    # Obtém os nomes das interfaces primeiro
    for i in interfaces:
        nomes.append(str(i))
    # Depois, imprimir os valores:
    for i in nomes:
        resposta = str(i) + ":" + '\n'
        for j in interfaces[i]:
            resposta = resposta + str(j) + '\n'
        resposta = resposta + '-----'
        lista.append(resposta)
    bytes = pickle.dumps(lista)
    socket_cliente.send(bytes)

# Fecha conexão do servidor
socket_servidor.close()
input("Pressione qualquer tecla para sair...") # Espera usuário ler
```

E com isso se encerra o projeto de bloco de Arquitetura de Computadores, Sistemas Operacionais e Redes.

Considerações finais:

No início tive algumas dificuldades para entender a estrutura de código que foi retirado do roteiro de aprendizagem das etapas 2, 3, 4, 5, 6, 7 e 8, porém consegui entender o que fazia cada linha do código. Com o entendimento, adicionei alguns outros itens nas estruturas de código para uma melhor visualização.

Referências:

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253423>

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253431>

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253439#>

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253439>

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253447>

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253455>

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253463>

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253471>