**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 sobra a arquitetura do processador, como o nome e modelo. Além disso, também é possível encontrar informações sobre o sistema operacional.

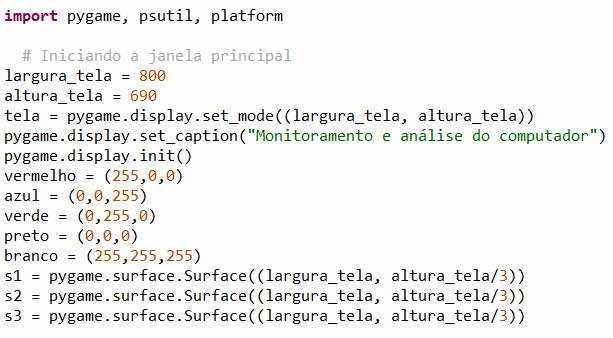
**CPUINFO**

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.

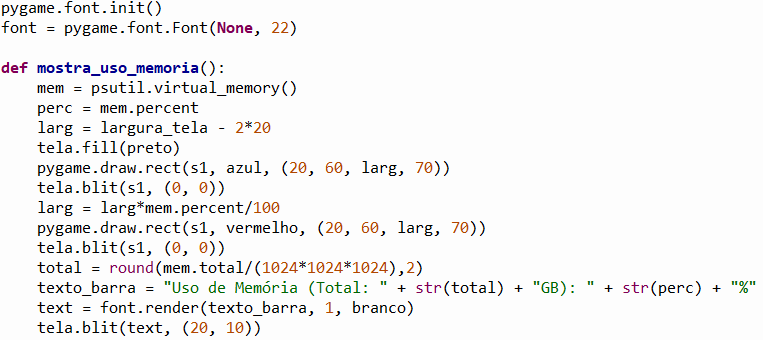
Para criar a interface gráfica, inicialmente foi gerado uma janela e uma estrutura de repetição até que o botão de sair seja pressionado.



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.



Essa função mostra a 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. O 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.

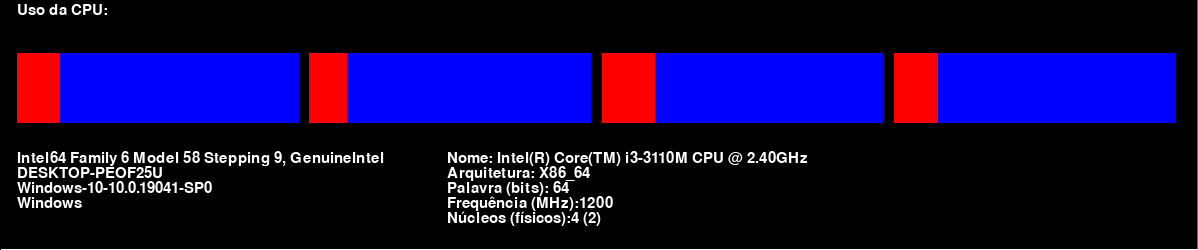
Além da arquitetura, o “cpuinfo” também indica o tamanho da palavra. A palavra é utilizada para indicar a unidade de transferência e processamento de um computador. As palavras são múltiplos de 1 byte, sendo que os microprocessadores geralmente utilizam 32bits – 4 bytes como tamanho da palavra (já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



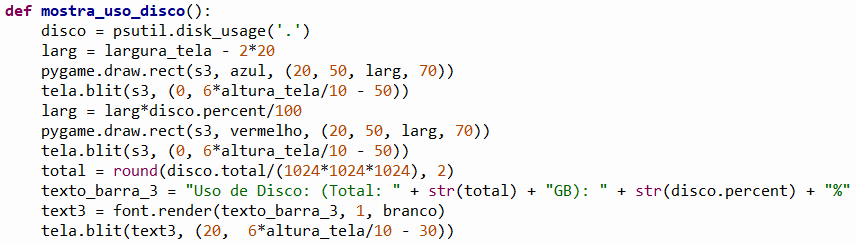
Note que, a estrutura de código da função “mostra\_uso\_CPU” foram colocados informações detalhadas do processamento através da biblioteca “platform”, foram colocados, também, informações detalhadas da CPU através das bibliotecas “cpuinfo”(nome, arquitetura e palavra) e “psutil”(frequência, núcleos lógicos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

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



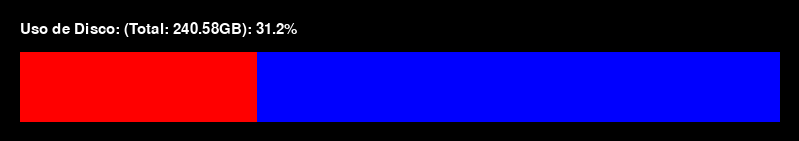
**Barra de uso de Disco**

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.



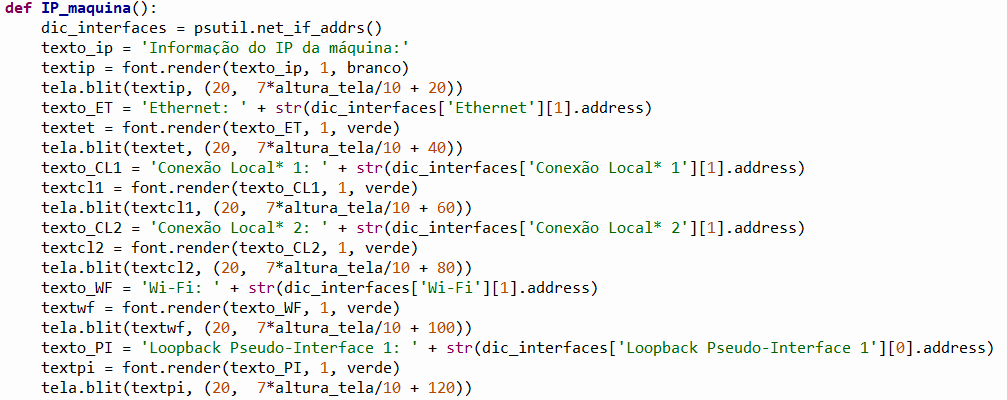
Note que a estrutura de código do uso de Disco é parecido com os códigos do uso de memória e do uso da CPU, porém com as informações específicas do Disco utilizando a função “psutil.disk\_usage(‘.’).

A barra de uso do disco ficou da seguinte forma:

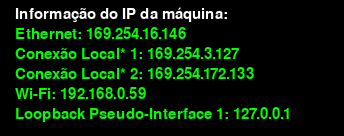


**Informações do IP da máquina:**

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.



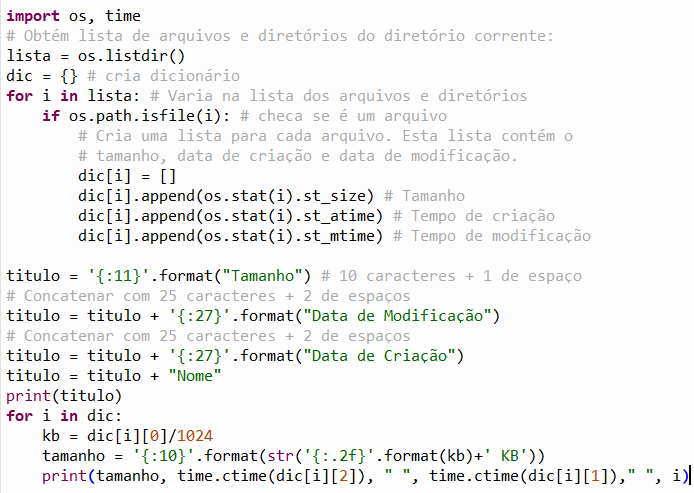
A exibição das informações do IP da máquina ficou da seguinte forma:



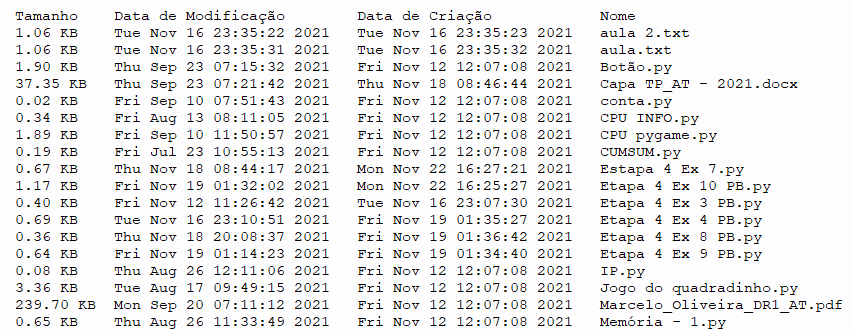
**Arquivos e diretórios**

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.

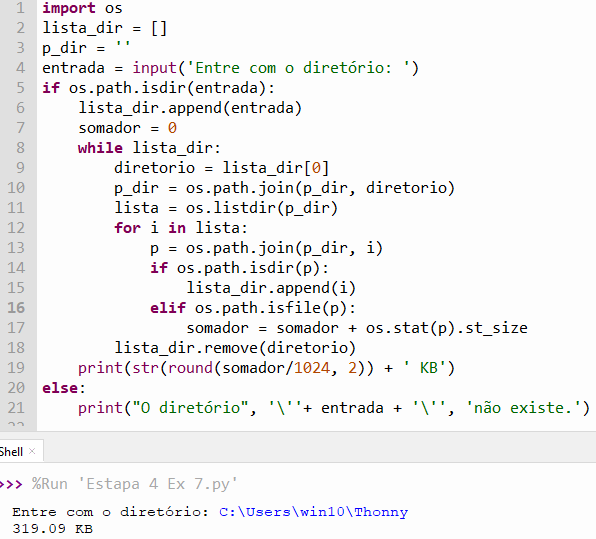
A primeira estrutura de código utilizada imprime os dados de todos os arquivos que estão dentro do diretório.



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.



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( )’.

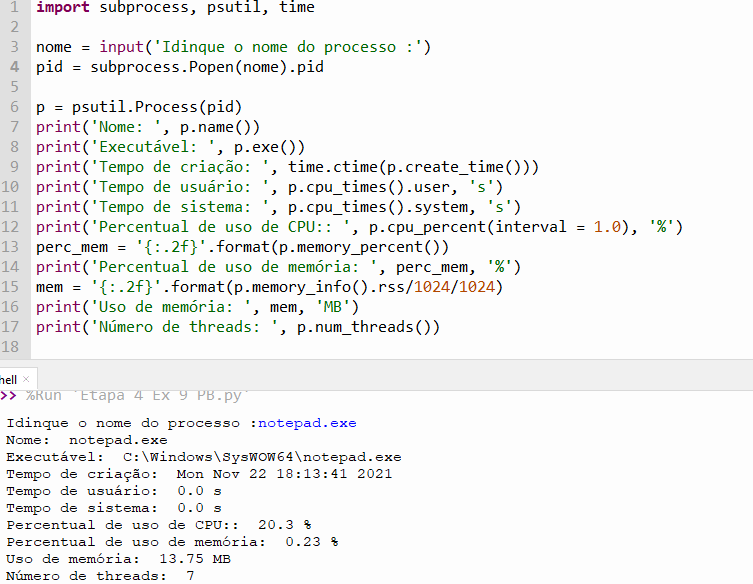


**Informações de Processo**

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

Foi criado uma estrutura de código pra listar a 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.

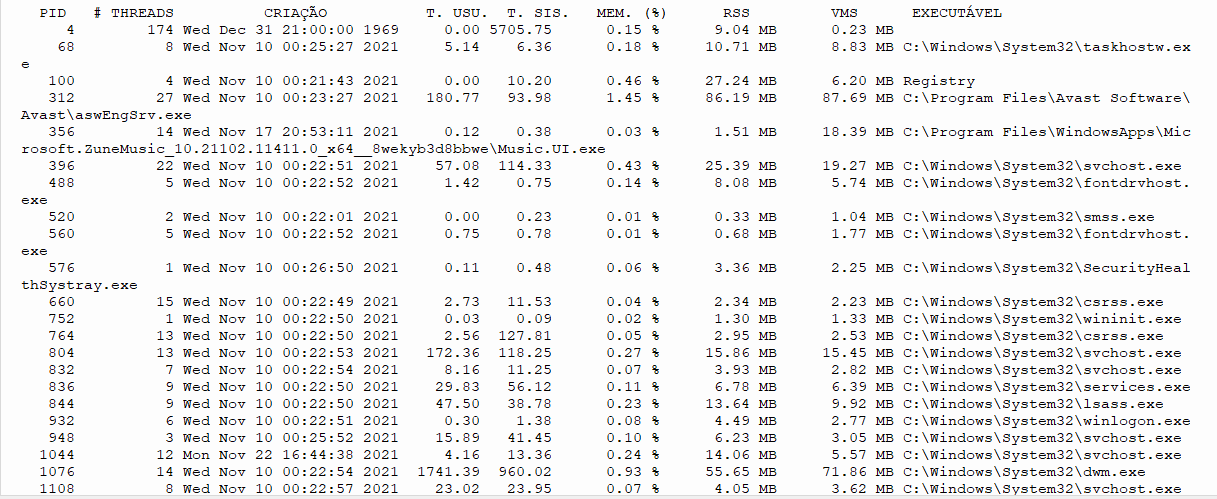
Para identificar o processo, foi utilizado o PID e para acessar as informações do processo, foi utilizado “psutil.Process(PID)”. O módulo “subprocess” foi utilizado para que o próprio processo criasse um outro processo que foi aquele monitorado.

****

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.



****

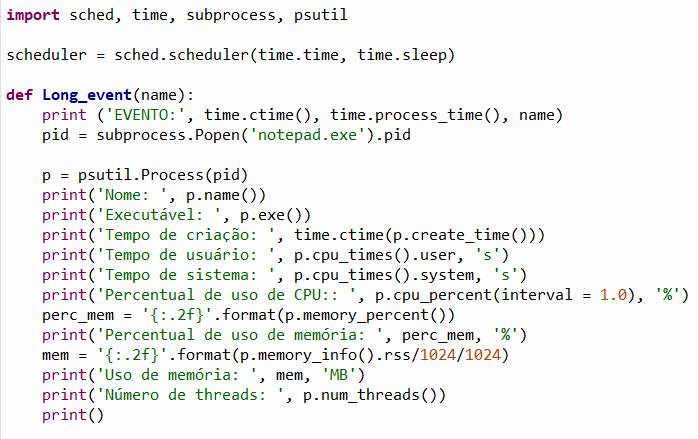
**Monitoramento e Gerenciamento de processos**

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

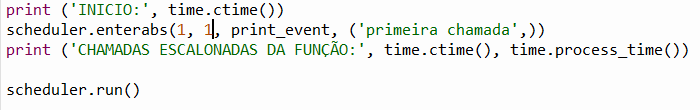
**Sched**

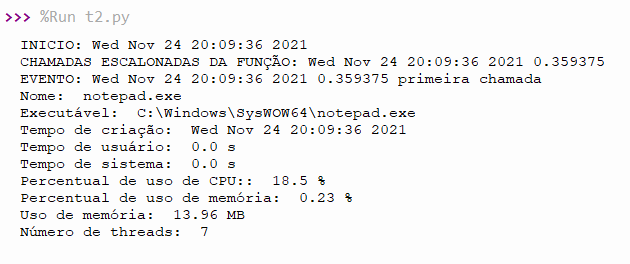
A biblioteca sched é utilizada para gerenciar a execução de processos, ou seja, utilizamos suas funções para escolher quando um processo vai ocorrer e quando vai ser interrompido.

Foi utilizado o módulo ‘sched’ para criar uma função que 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.

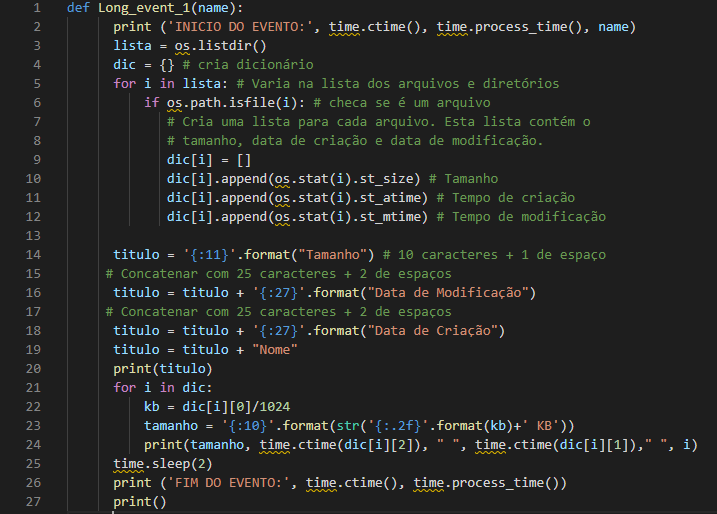


Após a criação dessa função, foi utilizado funções da biblioteca ‘sched’ pra poder chamar a função “Long\_event(name)”.

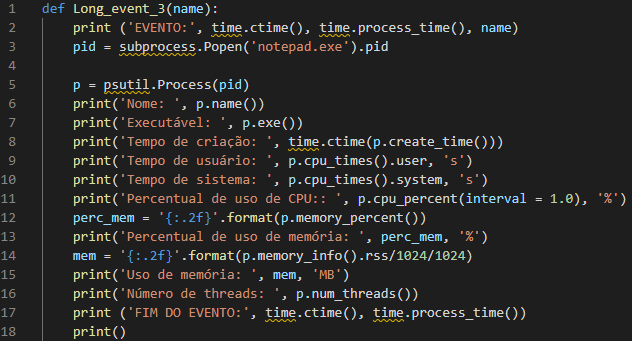




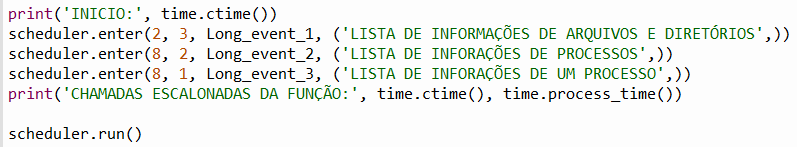
Para chamar mais de uma função, foi criado uma função de escalonamento de chamadas através de prioridade e delay. Foram utilizados 3 eventos para escalonar: lista de informações de arquivos e diretórios, lista de informações de processos, e lista de informações de um processo, respectivamente.



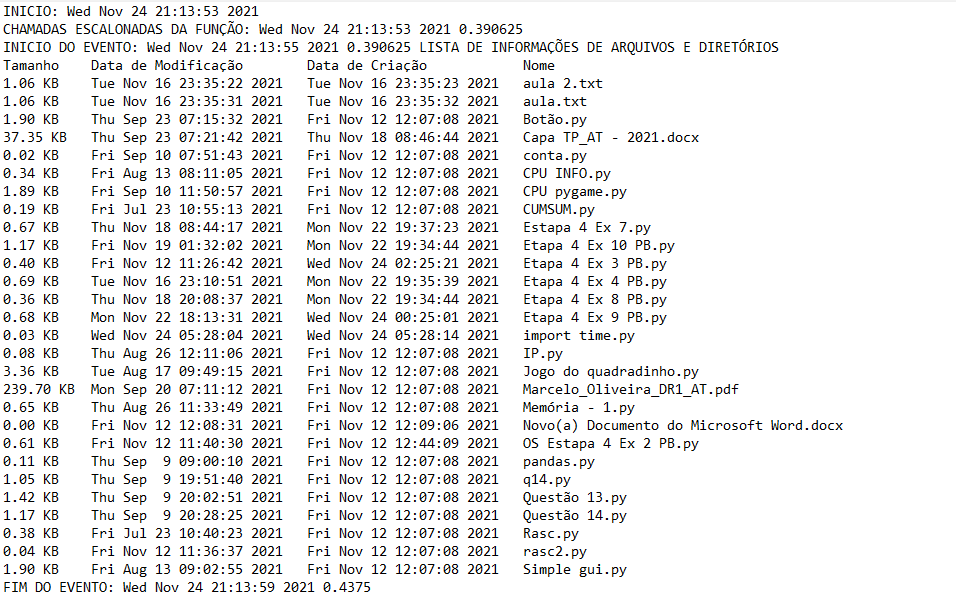
****

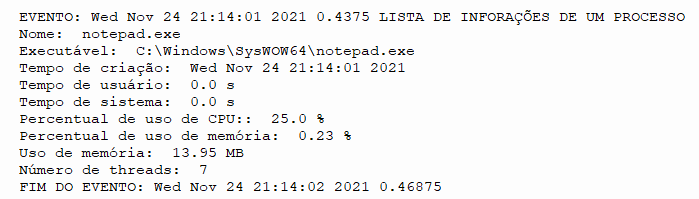
****

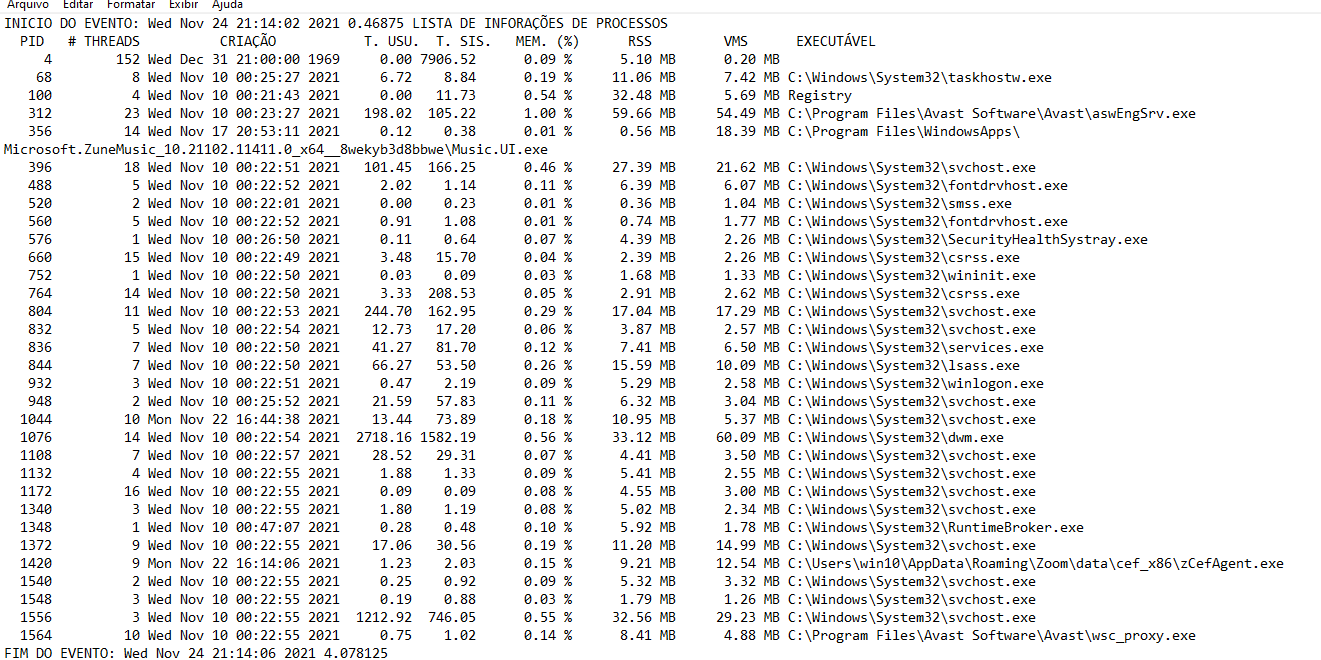
Foi escolhido um delay menor para o “Long\_event\_1” em relação ao “Long\_event\_2” e “Long\_event\_3”, que possuem o mesmo delay. Porém, o “Long\_event\_1” tem prioridade 3, “Long\_event\_2” prioridade 2 e “Long\_event\_3” prioridade 1.



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.







**Tempo real x clock da CPU**

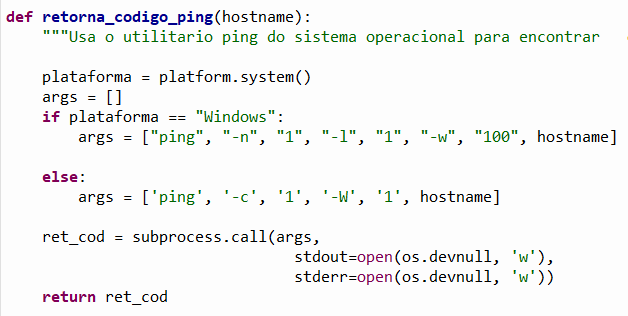
Quando as funções do “sched” foram executadas, foi apresentado, no início e no fim do evento, o tempo real e o clock da CPU. Existe uma diferença nesses dois tempos. O tempo real calcula o tempo de início e fim do evento. Já o clock calcula o tempo de processamento do evento. Isso pode ser notado quando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

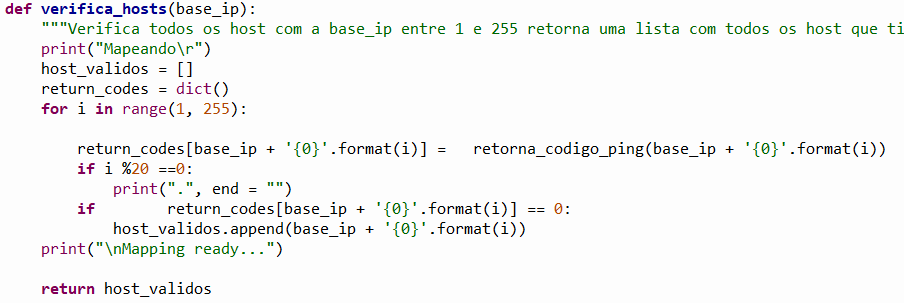
**Redes e Subredes**

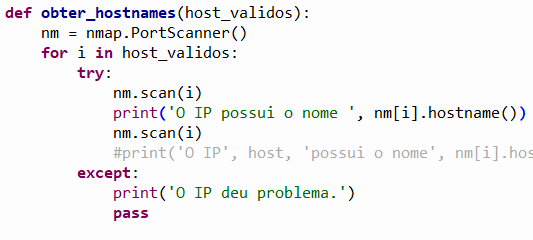
Nessa fase, foi criado funções para explorar as redes e subredes que se encontram as máquinas. Os módulos utilizados foram “nmap”, “subprocess”, “platform” e “os”.

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

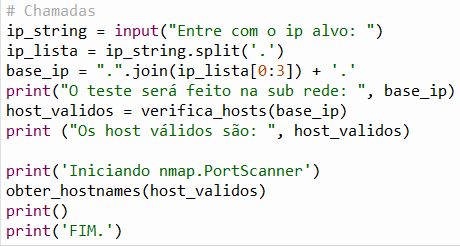
A primeira função é o “código\_retorna\_ping”, função essa que vai fazer a chamada do ping. Após, foi criado a função “verifica\_hosts”, que vai varrer a subrede para buscar outros IPs que estão na subrede e retorna uma lista com todos os IPs ativos na subrede. Por fim, a função “obter\_hostnames”, função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



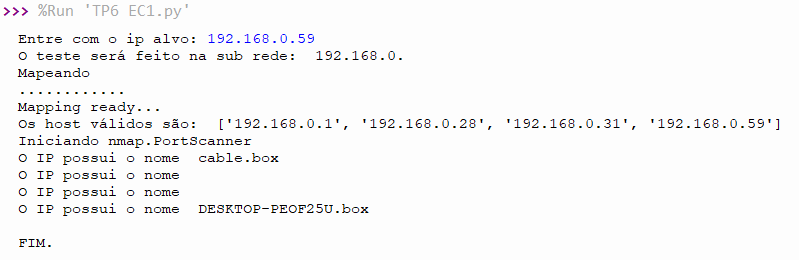




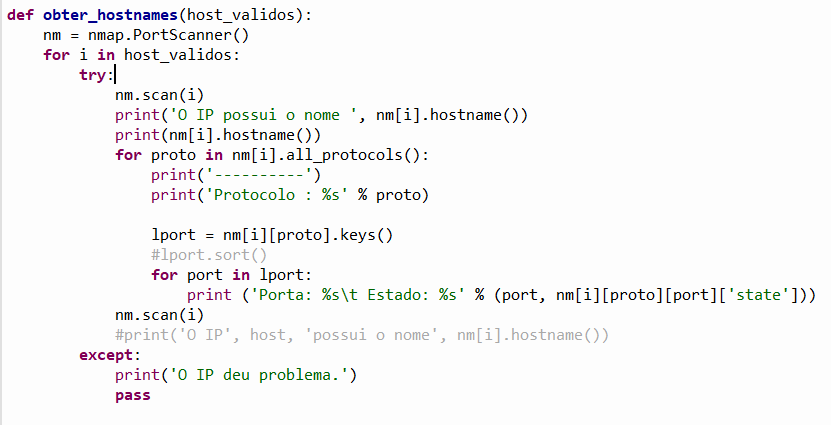
Após a criação da função foram feitas as chamadas.



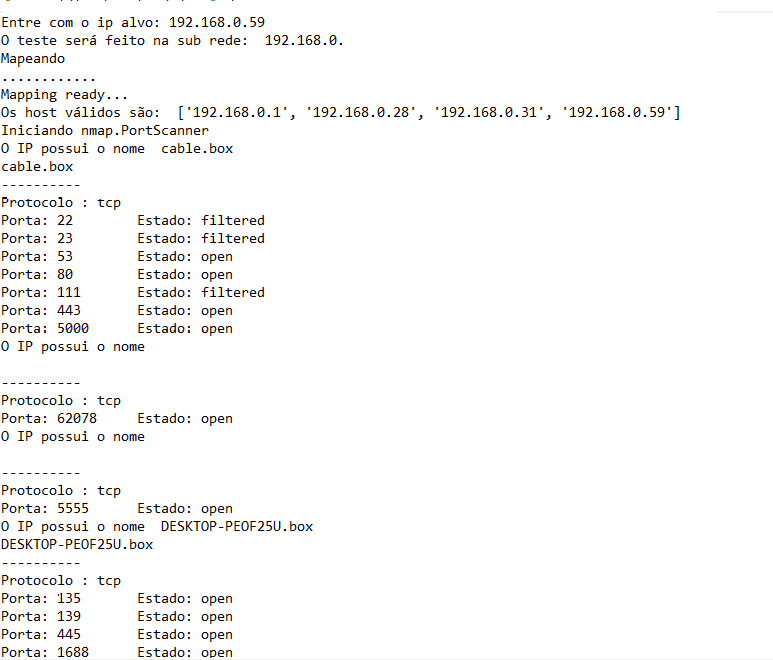
O resultado ficou assim:



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.



O resultado fica assim:



**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”.

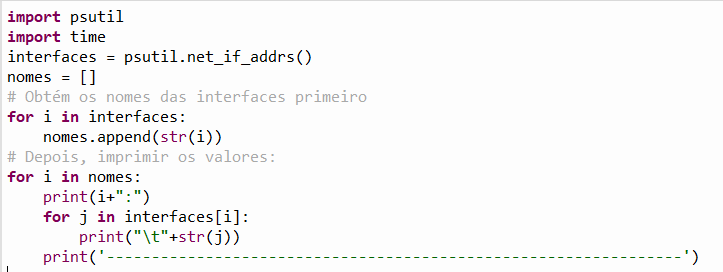
**Informações de Rede**

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.

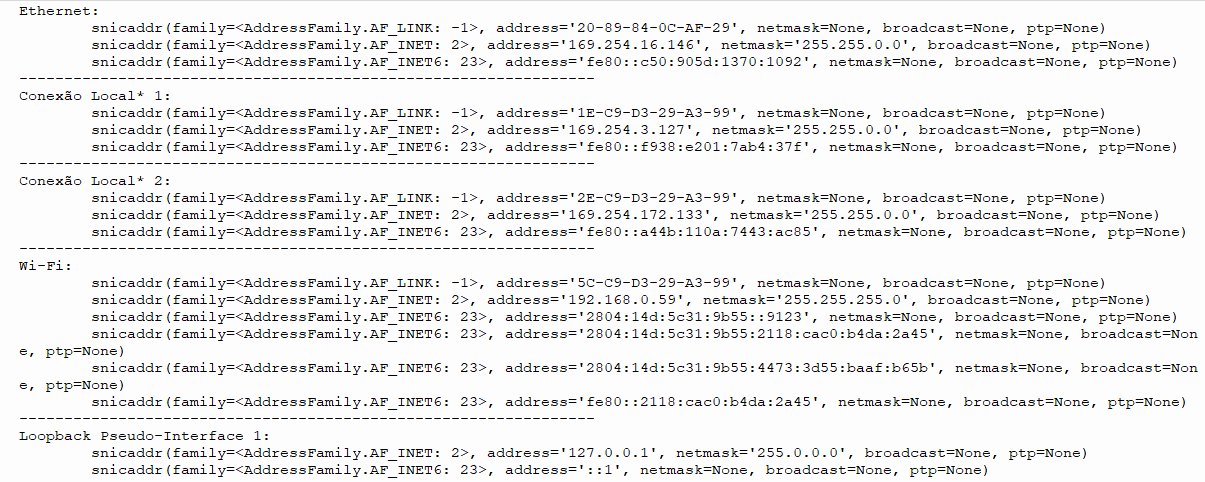
A primeira função do “psutil” utilizada foi “psutil.net\_if\_addrs()”. Essa função retorna as seguintes informações:

* **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:



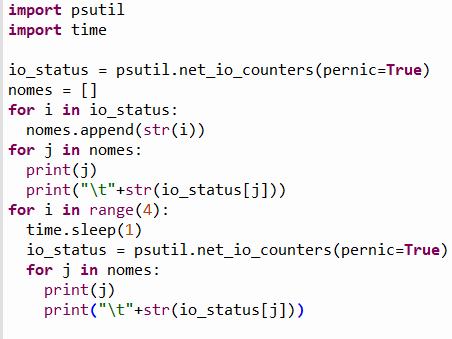
Essa estrutura retorna as seguintes informações:



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:



Resultado fica da seguinte forma:

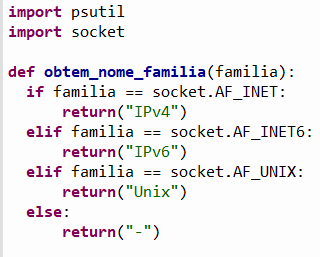


A última estrutura de código foi criada pra buscar informações de processos usando redes. Essas informações são:

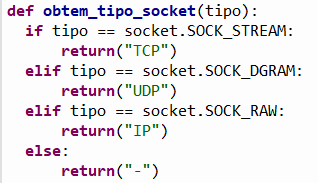
* **Tipo do endereço (End.)**: pode ser IPv4 ou IPv6 (ou Unix, para sistemas Unix).
* **Tipo de conexão (Tipo)**: pode ser TCP, UDP ou IP.
* **Status da conexão (Status)**: existem vários valores possíveis, como, por exemplo, ESTABLISHED, CLOSE, LISTEN, NONE.
* **Endereço local**: valor do endereço IP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

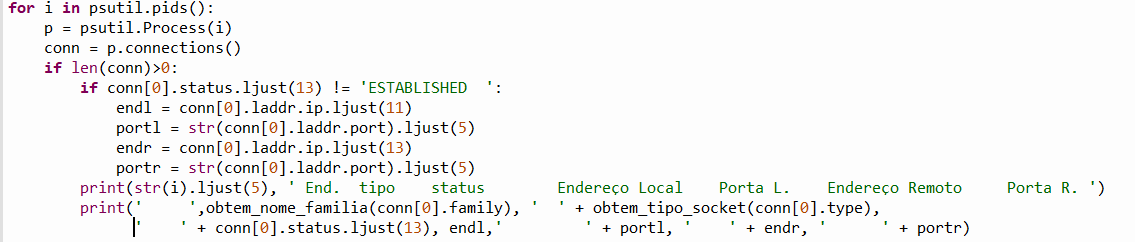
Nessa estrutura de código, foi utilizado, além do módulo “psutil”, o módulo “socket”. Do “psutil”, foi utilizado a função “psutil.pids()” para obter os PIDs dos processos dos sistemas. Além dele, foi utilizado o “psutil.Process()”. Do módulo “socket”, duas funções foram criadas: “obtem\_nome\_familia()”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

A função “obtem\_nome\_familia()” foi criada pra retornar o tipo de endereço.

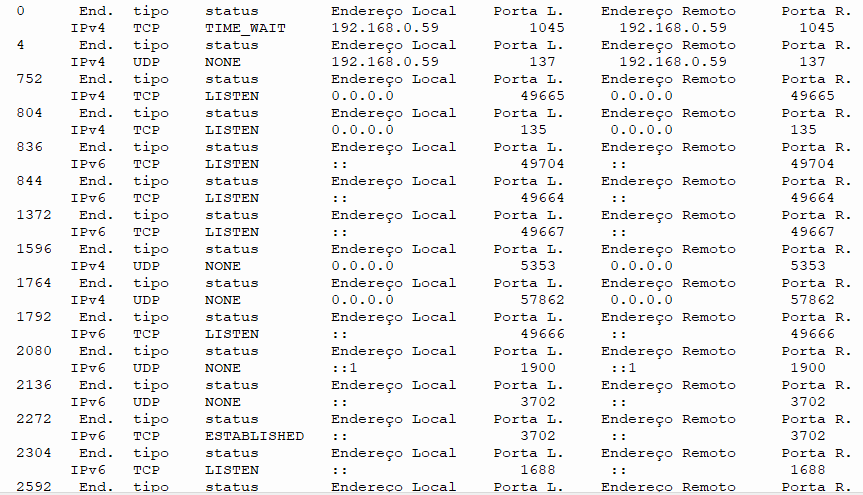


Já a função “obtem\_tipo\_socket()” foi criada pra retornar o tipo de conexão.





O resultado fica da seguinte forma:

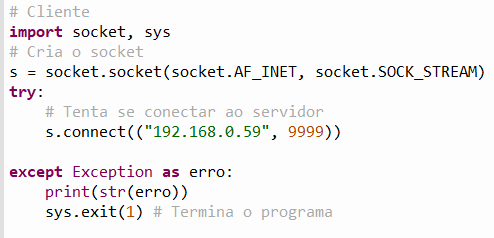


**Serviços de Rede**

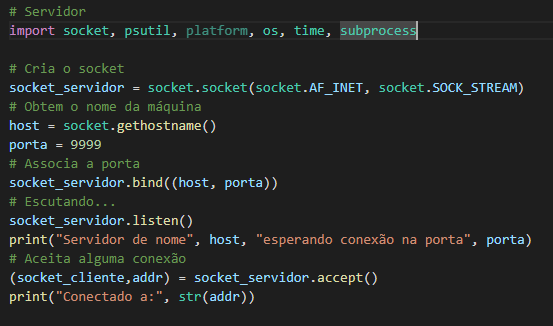
Nessa fase, 2 estruturas de código foram montadas para fazer o transporte de informações na rede através do serviço cliente-servidor. Para isso, foi utilizado o módulo “socket”. Além desse módulo, foi utilizado também outros como “psutil”, “platform”, “os”, “time”, “sys” e “subprocess” para buscar as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

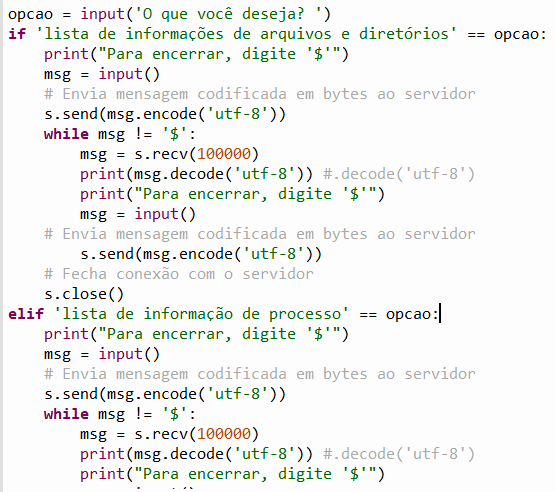


No modelo servidor, a estrutura de código para estabelecer a conexão da seguinte forma:

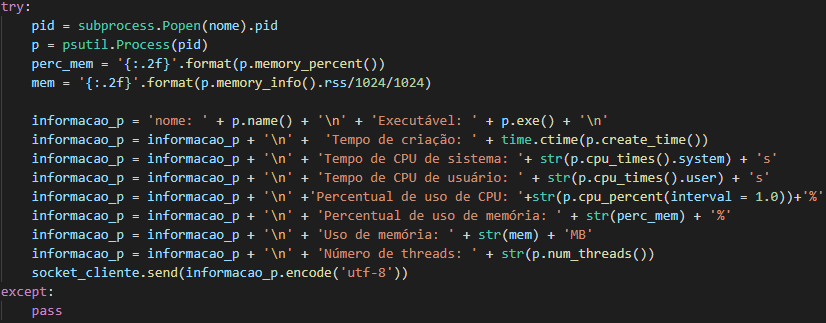


No modelo servidor, foi criado o socket, foi obtido o nome da máquina e a porta de acesso.

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.

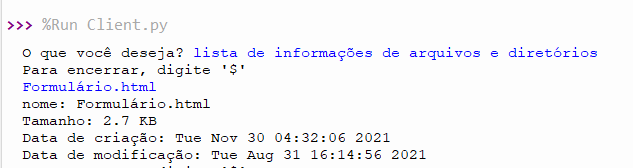




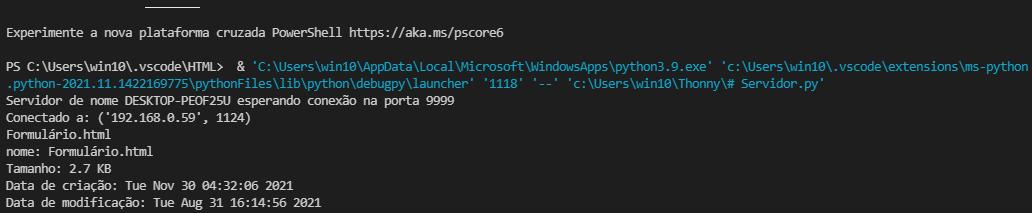
****

O resultado fica da seguinte forma:

Cliente



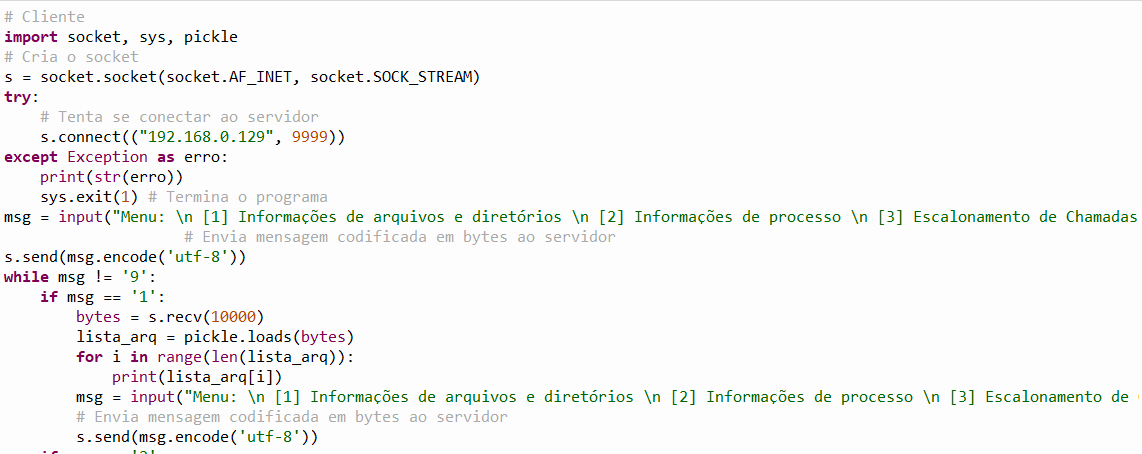
Servidor

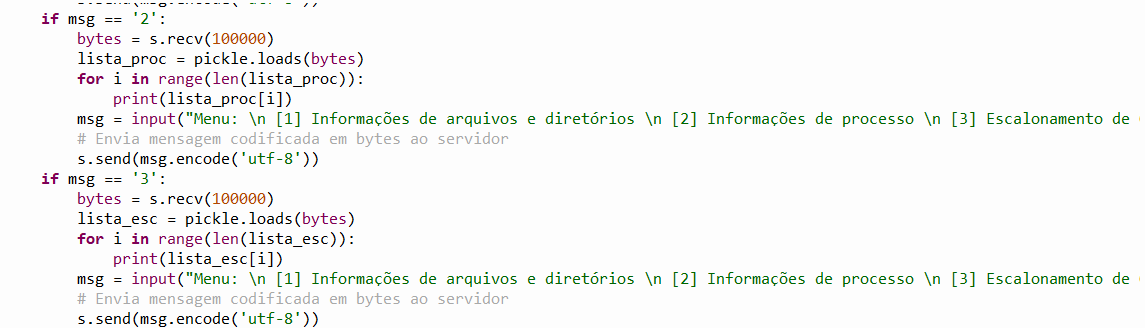


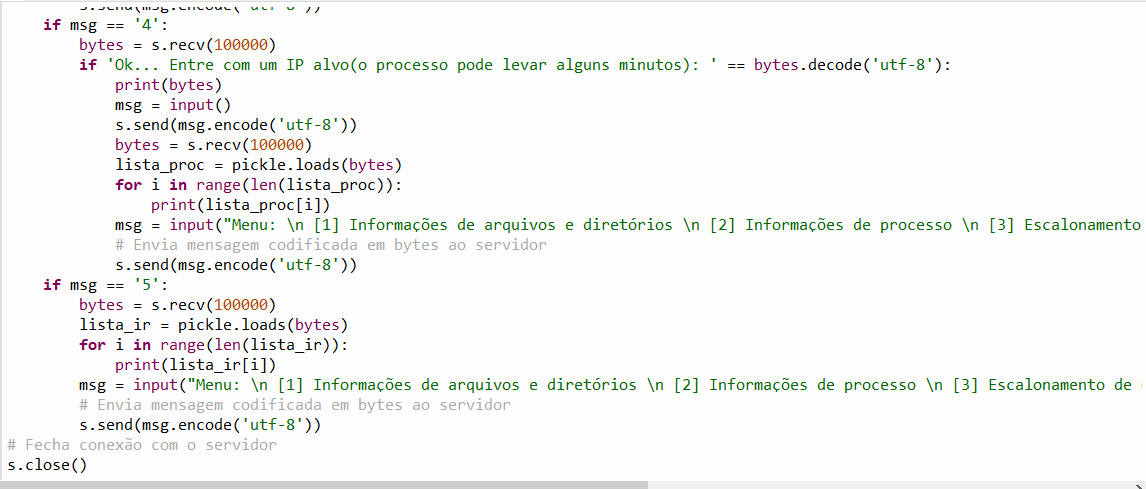
**Fase final:**

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.

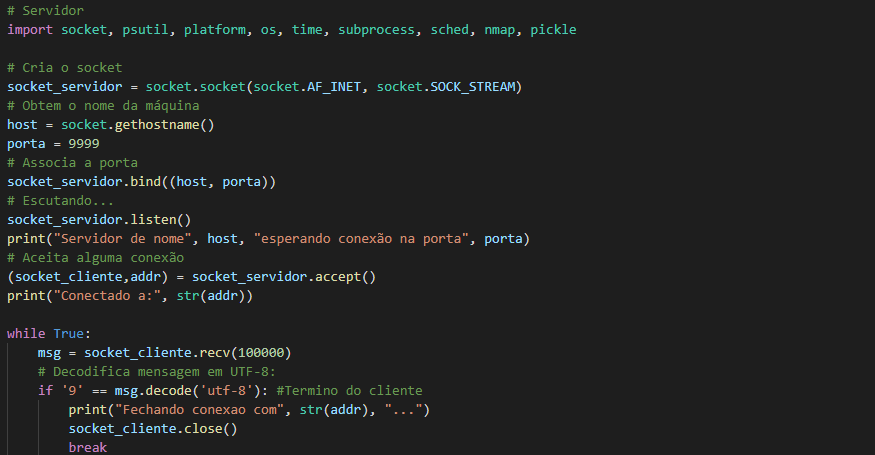
O código do cliente ficou da seguinte forma:

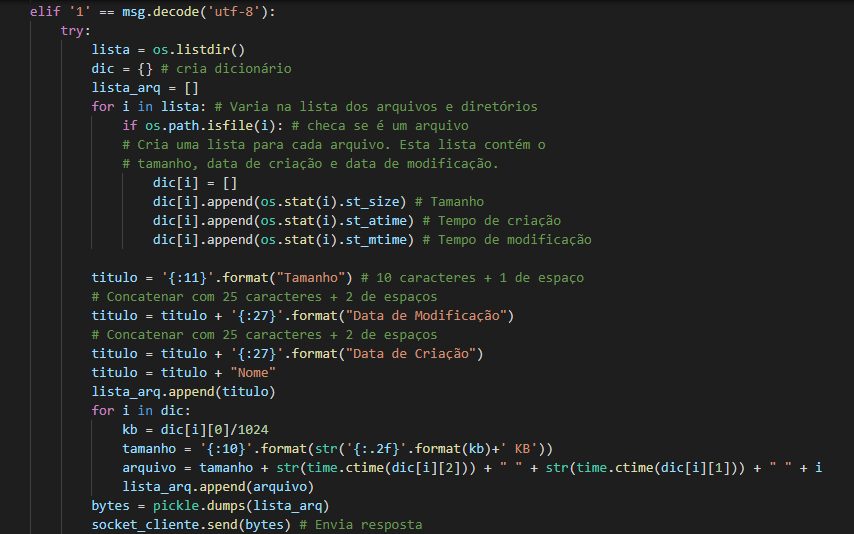






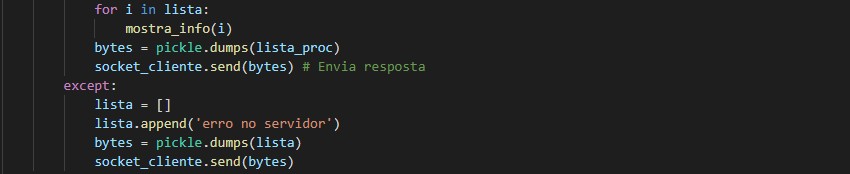
O código do servidor ficou da seguinte forma:



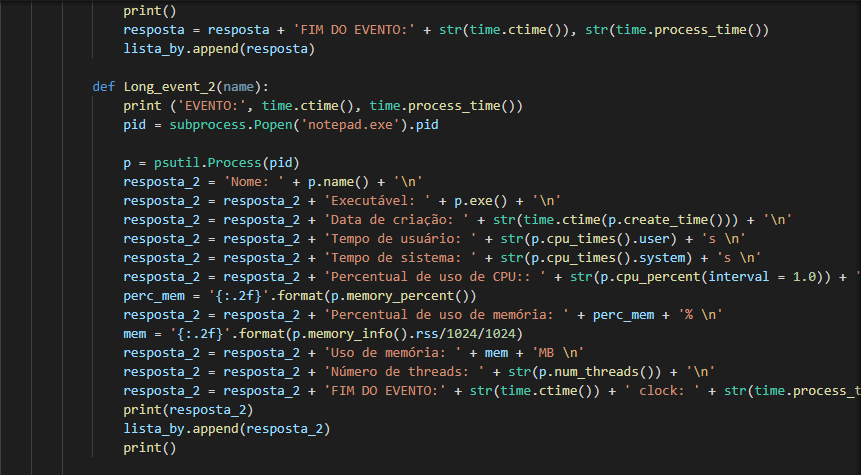


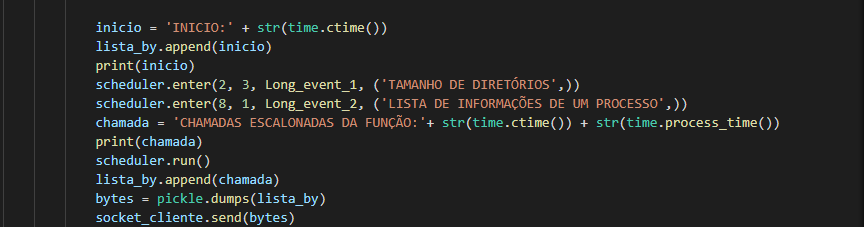


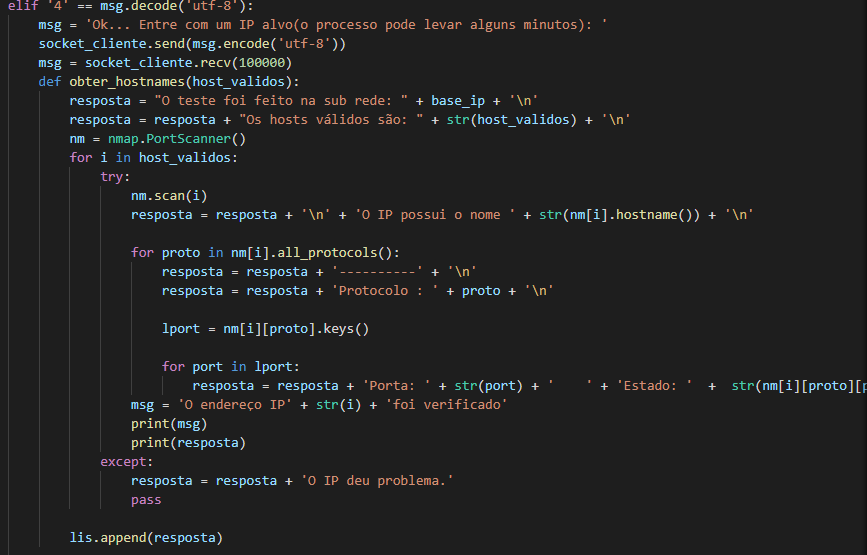


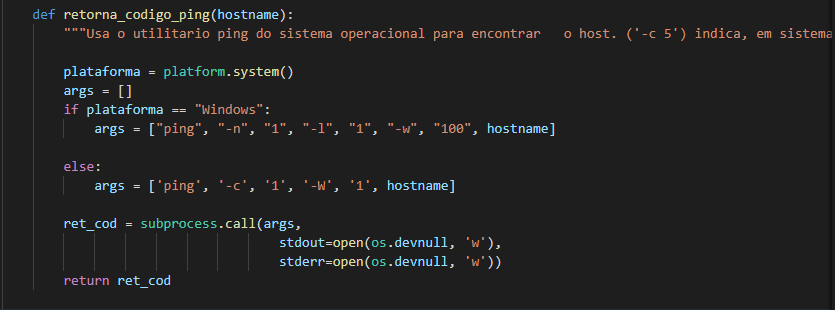


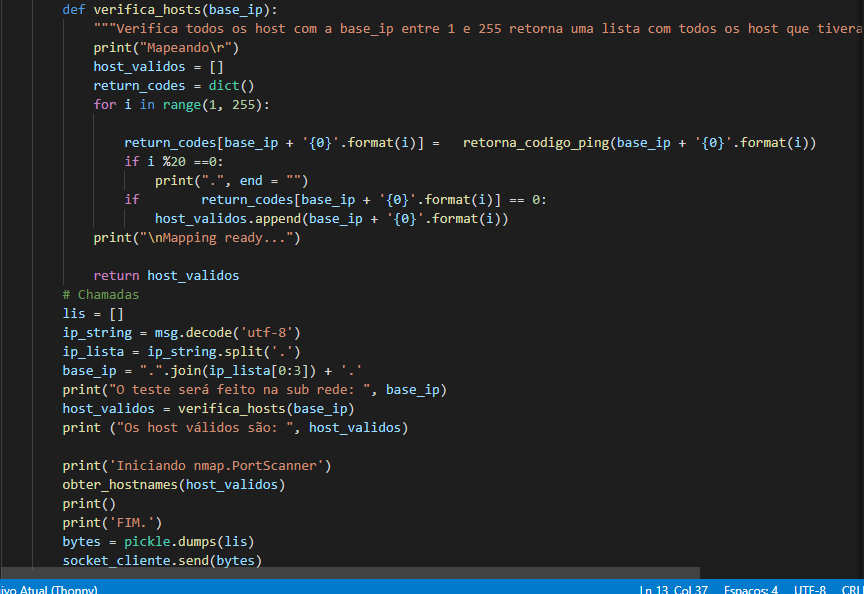


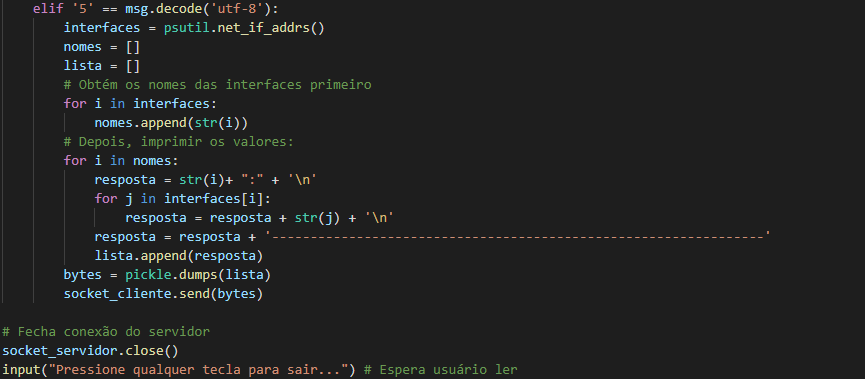






****

****

****

### 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=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