PROJETO SISTEMAS OPERACIONAIS
(OPÇÃO 2 – EXPERIMENTO REAL)
FORMULÁRIO INDIVIDUAL DA ATIVIDADE PRÁTICA

IDENTIFICAÇÃO	
NOME DO(A) ESTUDANTE 1: Maria Clara Pereira da Costa	
MATRÍCULA: 20211380040	**PERÍODO: 2º**

IDENTIFICAÇÃO	
NOME DO(A) ESTUDANTE 2: Italo Fernando Sousa Burity	
MATRÍCULA: 20211380015	**PERÍODO: 2°**

DESCRIÇÃO DO CENÁRIO

Resumo do projeto:

Linguagem utilizada:
- Python

Bibliotecas utilizadas:
- Os
- Time

O objetivo do projeto, é monitorar os recursos utilizados pelo Desktop em ambientes com o python instalado manualmente (Windows), e outro, com o python nativo do sistema operacional (Ubuntu). Os recursos de hardware disponibilizados serão os mesmos. Durante a execução de um programa em Python (SoBET), que define uma grande quantidade de jogadores e cartelas, onde é encerrado somente quando há um vencedor ou vencedores.

Outras observações sobre o "Sorteio":
- Os números sorteados pela cartela não são repetidos
- No código serão efetuados inicialmente dois testes, na primeira execução, será executado com 10 mil jogadores, e depois, com 100 mil.
- Será comparado entre Windows e Linux-Ubuntu.

Questão de Pesquisa:

- Qual o impacto no tempo de execução de um código, quando executado em dois sistemas operacionais diferentes, sendo um com python nativo (Linux - Ubuntu), e outro com python instalado manualmente (Windows).

1) DESCRIÇÃO DO AMBIENTE DE AVALIAÇÃO

Informações da memória:

Memória Windows

MemTotal: 5.5GB
MemFree: 3.6GB
MemAvailable: 5.5GB

Memória Ubuntu

MemTotal: 6078664 kB
MemFree: 4115580 kB
MemAvailable: 4859196 kB

Informações do processador

Processador Windows

Thread per core: 1
Model name: AMD Ryzen 7 3700U
CPU MHz: 2300

Processador Ubuntu

Thread per core: 1
Model name: AMD Ryzen 7 3700U
CPU MHz: 2299.998

2) CÓDIGO DO PROJETO

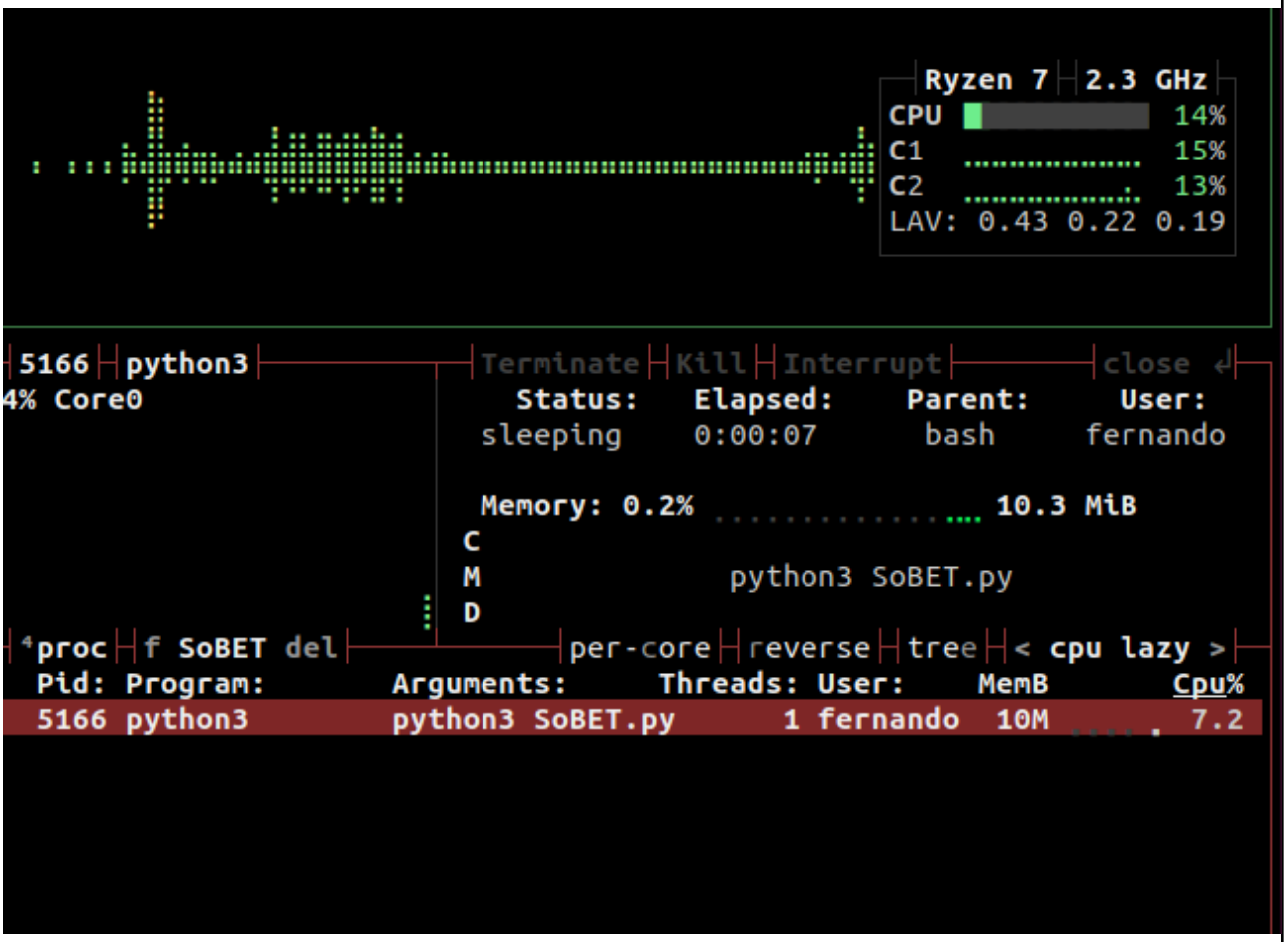
Link para visualizar execução:
https://colab.research.google.com/drive/12NqVyupH_MevWsRK8nUSpGt45EYAJGyT?usp=sharing

```
import os
from random import randint
import time

def contido(n, lista):
    if (n in lista):
        return True
    else:
        return False


def novo(n1, n2):
    numeros = []

    while (len(numeros) < 10):
        num = randint(n1, n2)
        if (contido(num, numeros) == False):
            numeros.append(num)
    return numeros


def numero_sorteado(n1, n2, nsort):
    while True:
        n = randint(n1, n2)
        if not contido(n, nsort):
            nsort.append(n)
            return n


def limpar():
    if os.name == 'nt':
        os.system('cls')
    else:
        os.system('clear')

start = time.perf_counter()
```

```
time.sleep(5)
num = (10000) #num (100000) para o segundo teste
cartelas = []
sorteado = []
ganhadores = []
vencedor = False
count = 0

qtde = 0
for i in range(num):
   lista = novo(1, 50)
   cartelas.append(lista)

while not vencedor:
   sort = numero_sorteado(1, 50, sorteado)
   print("SoBET")
   print(f"Número sorteado: {sort}")
   for i in range(len(cartelas)):
       qtde = 0
       for j in range(len(cartelas[i])):
           if (contido(cartelas[i][j], sorteado) == True):
               qtde += 1
       print(f"Jogador {i + 1}", (cartelas[i]), qtde)
       if qtde == 10:
           vencedor = True
           ganhadores.append(i)
   if not vencedor:
       print("Ainda não temos vencedores.")
       count = count + 1
       print('A quantidade de números sorteados até o momento é:
',count)
       time.sleep(1)
       limpar()

print()
for s in ganhadores:
   print(f"Jogador {s + 1} completou a cartela! ", cartelas[s])

print("Números sorteados: ", sorteado)

end = time.perf_counter()
print(end - start, "Segundos")
```

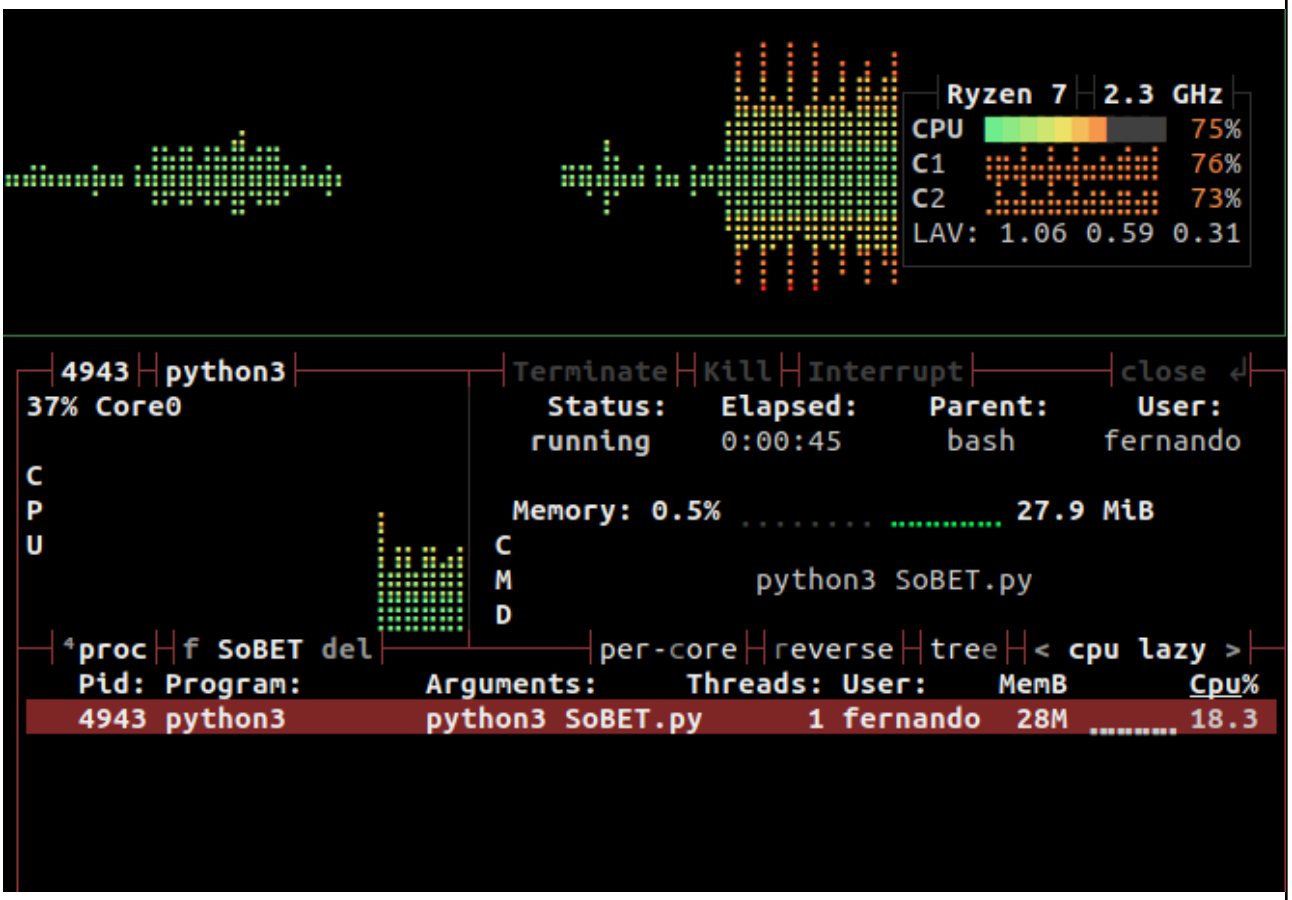
3) RESPOSTA

LINUX - UBUNTU - Execução com 10.000 jogadores:



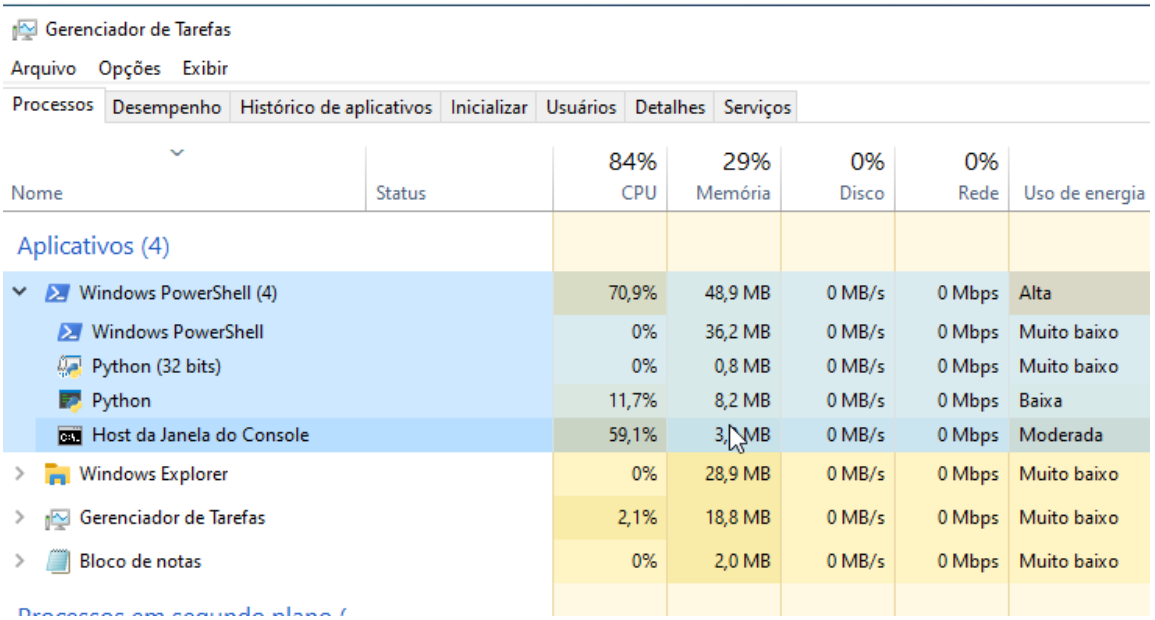
- Ao executar o programa com 10.000 jogadores, utilizando a ferramenta “bpytop” para monitorar e foi verificado o consumo em torno de 7.2% da CPU inicialmente, utilizando apenas 0.2% da RAM.

LINUX - UBUNTU com pico de jogadores em 100.000:

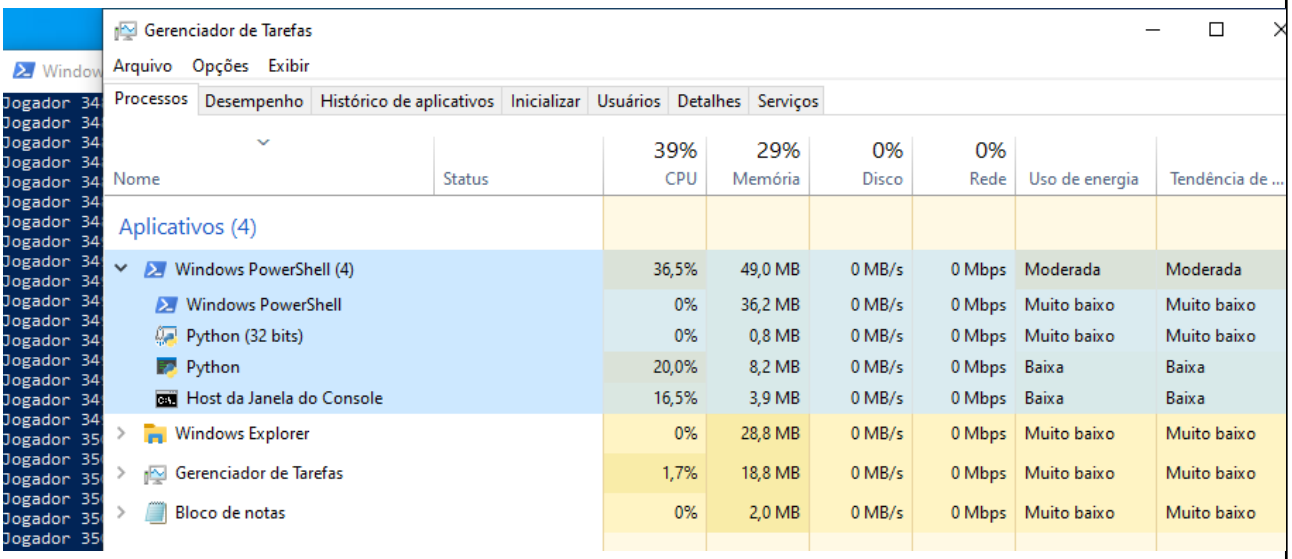


- Na execução do programa contendo 100.000 jogadores, já é possível observar o consumo maior do recurso da CPU. Chegando a consumir 18.3%, esse valor foi variando para baixo em alguns momentos, mas sempre próximo desse uso. O consumo de RAM subiu em relação anterior apenas para 0.5%

WINDOWS - Execução com 10.000 jogadores:

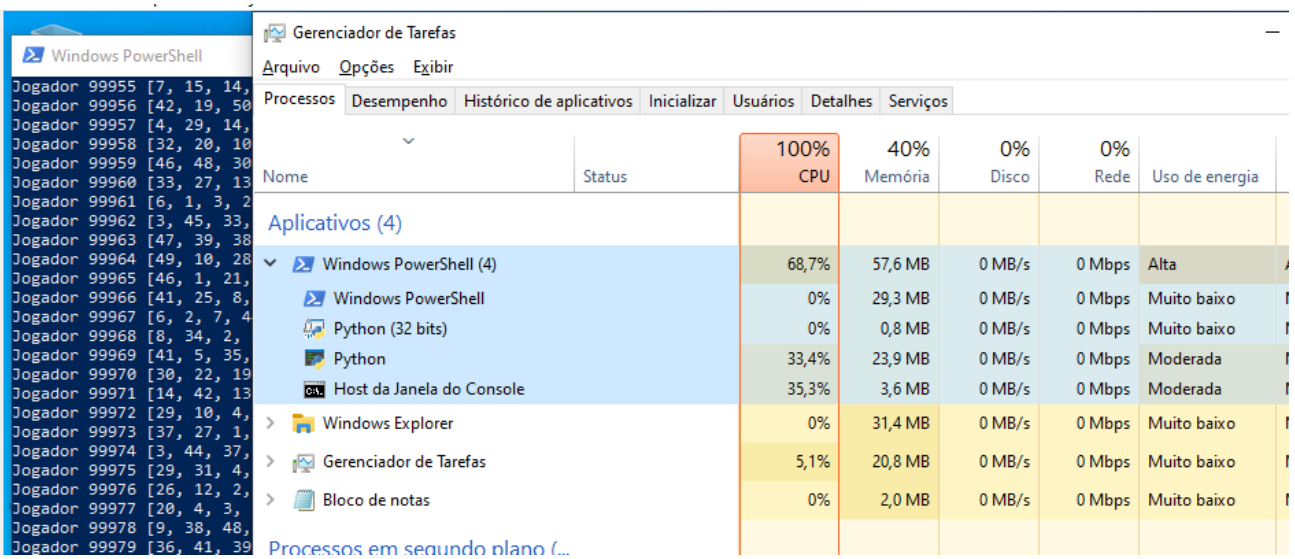


- Quando o programa iniciou, foi possível observar um pico no uso da CPU pelo processo.

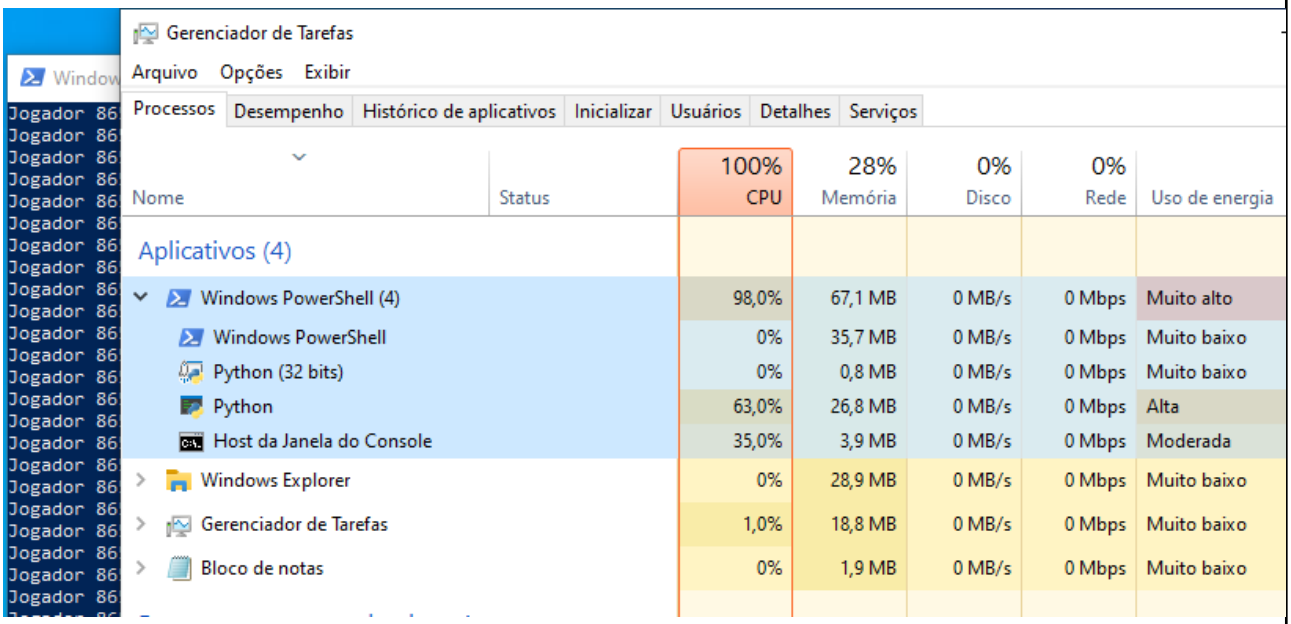


- 36,5% de USO da CPU pelo PowerShell enquanto o programa com 10 mil jogadores estava sendo executado após o pico inicial. Esse valor sofreu diversas variações, mas sempre em uma média próxima ao valor apresentado na imagem. O consumo de memória RAM foi de 49,0MB, superior ao consumo pelo LINUX.

WINDOWS - Execução com pico de jogadores em 100.000:

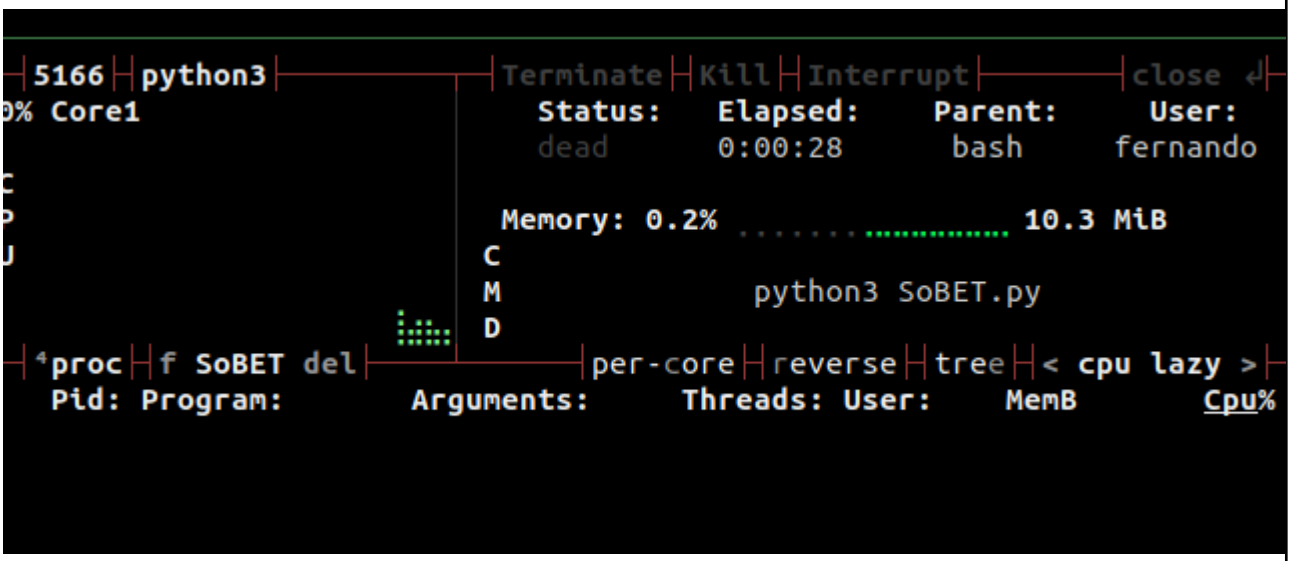


- Ao executar o programa com 100.000 jogadores. O consumo da CPU aumentou consideravelmente. Antes saindo de um valor médio que variou na faixa dos 30%, foi possível verificar uma média girando em torno dos 60% de consumo da CPU.

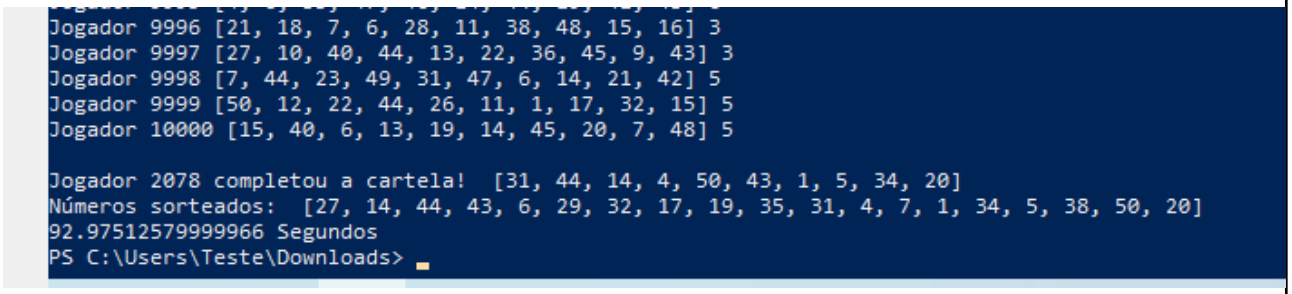


- Em alguns momentos foi possível visualizar o pico de consumo de 98% da CPU, mas esse valor não permaneceu por muito tempo, retornando aos valores médios que foram apresentados.

Comparação do tempo de execução com 10.000 jogadores.

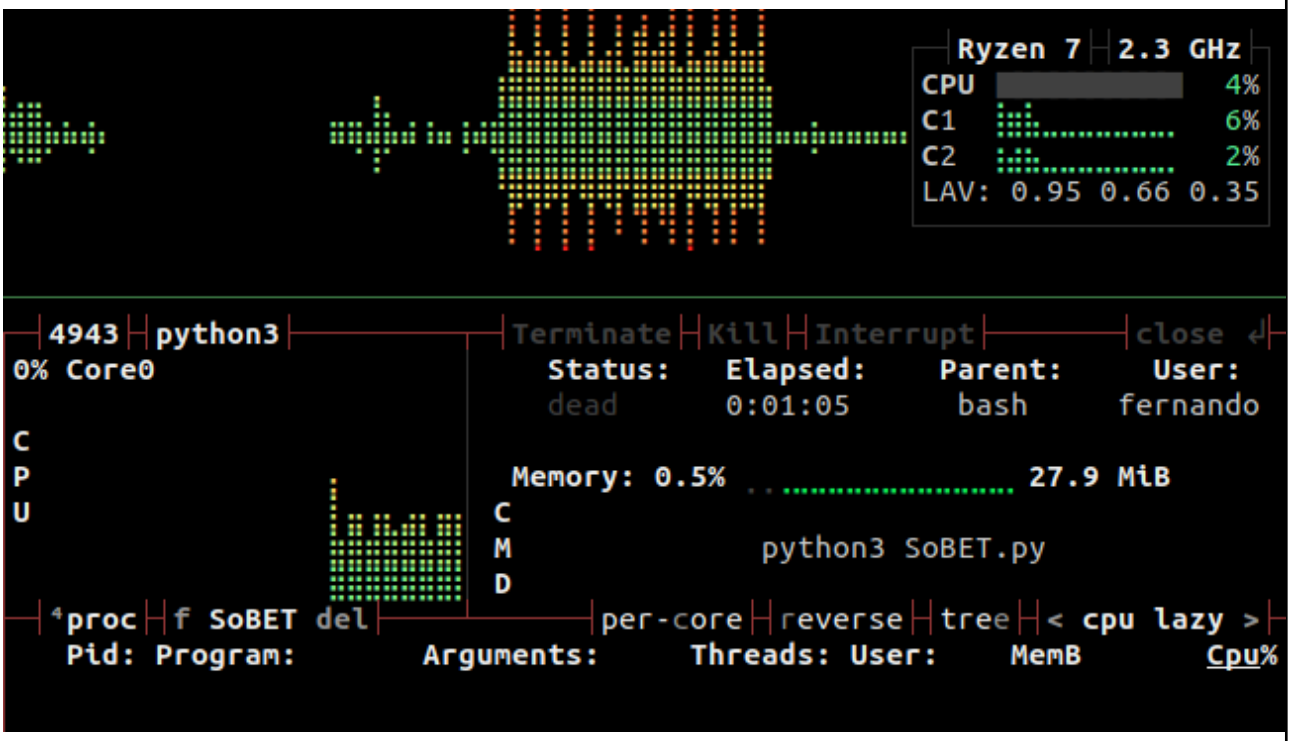


- No Ubuntu o programa com 10.000 jogadores foi executado e finalizado em torno de 28 segundos.

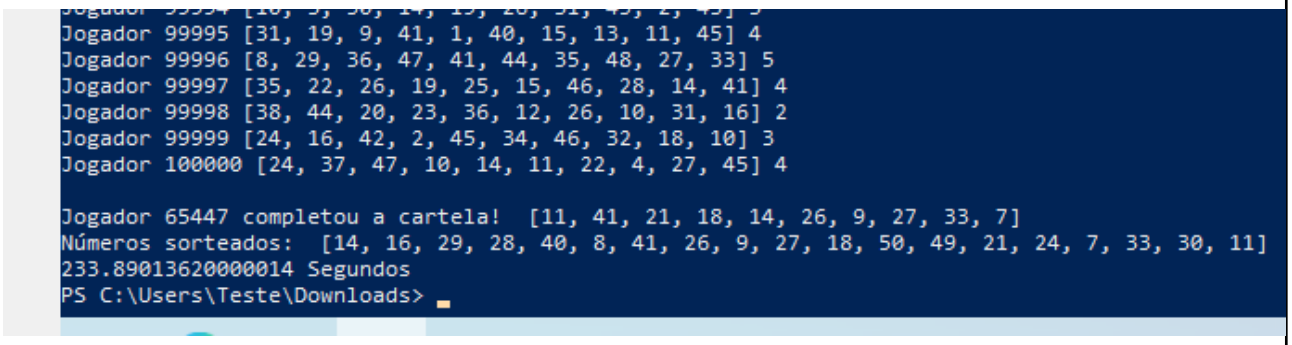


- No Windows o programa com 10.000 jogadores foi executado e finalizado em torno de 92 segundos. (OBS. Foi usando um comando em Python para medir o tempo utilizado ao executar o código no Windows, o comando em questão foi o “time.perf_counter()” da biblioteca time.)

Comparação do tempo de execução com 100.000 jogadores.



- No Ubuntu o programa com 100.000 jogadores foi executado e finalizado em torno de 1 minuto e 05 segundos.



- No Windows o programa com 100.000 jogadores foi executado e finalizado em 233 segundos, quase 4 minutos.

4) RESPONDER

A pesquisa se baseou em testar o comportamento de um mesmo programa python em dois ambientes diferentes. Foi tentado ao máximo igualar a quantidade de recursos direcionados para os ambientes em questão, para que a comparação fosse a mais próxima possível.

Como a proposta era visualizar o comportamento do processo em um ambiente que não possui o python nativo em seu sistema, o Windows foi a escolha e junto com ele, foi necessário instalar o python3 para que o teste pudesse ocorrer. Na tentativa de ter uma execução mais semelhante ao Linux, foi utilizado o powershell para que o código fosse executado e acompanhado com mais precisão.

Como mostrado nos resultados, o programa, em ambas as condições 10.000 e 100.000 jogadores, executado em um ambiente Linux, teve o uso de recursos bem inferior ao ambiente Windows. Foi utilizado o comando time.sleep() para que o intervalo entre o sorteio do programa fosse um pouco maior, para que a observação do comportamento do processo fosse mais detalhada. Com isso foi possível ver que naturalmente um Script em Python, executado em escalas menores ou maiores, são executados e utilizam menos recursos em ambientes como o Linux-Ubuntu que possuem o python nativo em seu sistema. Acrescentando um adendo ao tempo de execução, que se mostrou bem menor no ambiente Linux, comparando a execução dos dois processos em 100.000 jogadores, vemos a diferença de 168 segundos, com o Linux finalizando o programa em 65 segundos e o Windows em torno de 233.

Em suma. Os dois sistemas operacionais conseguiram executar o programa e manter sua execução até o final. Foram usadas ferramentas como o Powershell + Gerenciador de Tarefas + python3, para que a verificação no ambiente Windows fosse realizada, testes que apresentaram certo resultado ao identificar a utilização de recursos pela máquina. Claramente esse programa sendo executado em uma máquina que tivesse mais recursos disponíveis, como mais Núcleos ou mais Memória RAM, obteríamos resultados melhores, mas como a questão era a comparar com um ambiente Linux que possui as mesmas especificações, apenas utilizando o programa bpytop + python3 nativo do sistema, foi possível comprovar o comportamento mais "fluído", onde no monitoramento o sistema apresentou uma execução com menos recursos e bem mais rápida pelo ambiente Linux-Ubuntu.