Infnet

Engenharia de Software

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

Rodrigo Pinto Coelho Gomes
Prof: Adriano Saad

Ultima atualizacao : 04/10/2020

SUMÁRIO

1. INTRODUÇÃO

O projeto “Performance Computer - PC”, nome por mim atribuído ao projeto, foi idealizado, visando atender requisitos da disciplina de Projeto de Bloco do Curso de Engenharia de Software da INFNET e teve seu início em julho/ 2020.

O principal objetivo deste projeto é a construção de um software cliente-servidor em Python, que explore os conceitos de arquitetura de redes, arquitetura de computadores e/ou de sistemas operacionais, acompanhado de relatório explicativo.

O desenvolvimento do projeto, se dará em etapas, visando facilitar testes e confecção da respectiva documentação, evoluindo somente com aprovação/testes da etapa imediatamente anterior, onde o somatório das etapas resultará na versão final do projeto.

A linguagem de desenvolvimento do software será o Python, versão 3.8.

A evolução e desenvolvimento do projeto, será mostrada através de versões, que conterão as alterações e melhorias, até atingirmos a versão final do projeto. Por definição começaremos a primeira versão do software como “VERSAO 2.0”

2. FUNCIONALIDADES

As principais funcionalidade do projeto até a etapa 5, são :

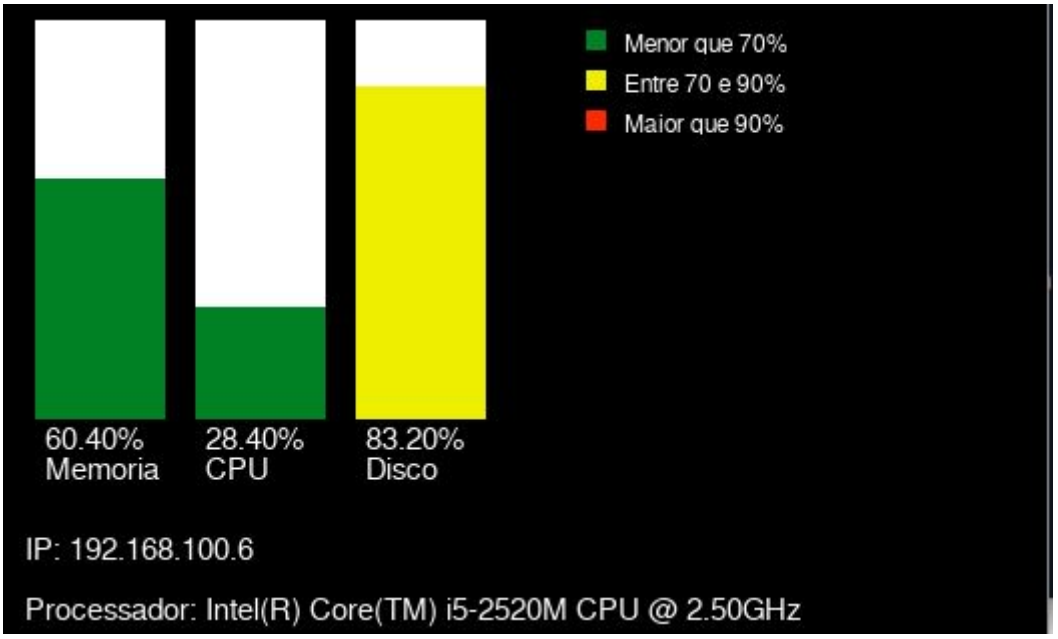
a) Uma barra de indicação da porcentagem do uso de memória;
b) Uma barra de indicação da porcentagem do uso de CPU, junto com a informação detalhada da plataforma de processamento;
c) Uma barra de indicação da porcentagem do uso de disco;
d) Um texto com a informação do IP da máquina.
e) Informações associadas ao processador
f) Informações associadas a memória
g) Informações associadas ao processado
h) Informações associadas ao Disco
i) Informações associadas ao IP
j) Rolagem de telas caso o usuário clique nas setas esquerda ou direita do teclado. Seguindo sempre uma ordem predefinida, como em um carrossel.
k) Um resumo de todas elas. O qual seria acessado quando o usuário clica na tecla “Barra” do teclado.
l) Barras de CPU associadas a cada núcleo (*core*);
m) Informação de nome/modelo da CPU (*brand*);
n) Informação do tipo da arquitetura (*arch*);
o) Informação da palavra do processador (*bits*);
p) Informação sobre a frequência total e frequência de uso da CPU;
q) Informação do número total de núcleos (núcleo físico) e threads (núcleo lógico).
r) Funções de informações sobre diretórios e arquivos, tamanho, localização, data de criação, data de modificação, tipo, etc.

s) Função para mostrar o resultado, sendo o resultado mostrado em texto formatado impresso na tela ou gráfico, usando o módulo ‘pygame’.
t) Funções que retornem informações sobre processos do sistema.
u) Módulo ‘sched’ para retorno de informações sobre diretórios e arquivos.
v) Escalonamento das chamadas das funções com o módulo ‘sched’ e medir o tempo total utilizado por cada chamada com o módulo ‘time’.
w) No escalonamento realizar uma comparação dos tempos obtidos com a quantidade total de clocks utilizados pela CPU para a realização dessas mesmas chamadas.
x) Diferença de tempo real e tempo do clock do computador.
y) Utilização do ‘time.sleep’ dentro de alguma chamada.

3. APRESENTAÇÃO VERSÃO 2.0

3.1 - Inicialmente foi criado uma superfície, na cor preta, que servirá de fundo, com objetivo de ressaltar as respostas do software. Nesta primeira superfície, temos 3 barras, com objetivo de informar instantaneamente os percentuais de memória, utilização de CPU e utilização de Disco Rígido. Adotamos níveis, com as cores verde, amarelo e vermelho para barras verticais, visando a rápida interpretação das respostas pelos usuários so software. Estes níveis foram divididos, conforme utilização, em : verde, menor que 90% de utilização , amarelo, entre 70% e 90% de utilização e vermelho, quando a utilização é maior que 90%, conforme legenda apresentada na tela.

Além disso, informações sobre o IP utilizado na conexão de internet e as características do detalhadas do processador, como o tipo de CPU, clock, conforme figura abaixo :



3.2 Utilizamos os módulos e ferramentas disponíveis na versão do Python 3.8, conforme descrição abaixo :

O **Pygame** é uma biblioteca criada para jogos multiplataformas, que tem como principal característica, se abster da lógica principal do software, sendo o uso de memória e CPU, tratados pelo próprio código do pygame.

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

A biblioteca **Plataform** é usado para recuperar o máximo possível de informações sobre a plataforma na qual o programa está sendo executado. Agora, por informações de plataforma, significa informações sobre o dispositivo, seu sistema operacional, nó, versão do sistema operacional, versão do Python, etc. Este módulo desempenha um papel crucial quando você deseja verificar se o seu programa é compatível com a versão do python instalada em um sistema específico ou se as especificações de hardware atendem aos requisitos de seu programa.

A biblioteca **Time** fornece muitas maneiras de representar o tempo no código, como objetos, números e strings. Ele também fornece outras funcionalidades além de representar o tempo , como esperar durante a execução do código e medir a eficiência do seu código.

As formas de importação seguem o padrão abaixo :

```
import platform
import psutil
import pygame, sys
from pygame.locals import *
import time
```

3.3 Utilizamos a biblioteca Psutil para obter as informações da memória, CPU e Disco. A forma para se obter tais informações, basicamente é importar a biblioteca Psutil, instanciando os respectivos métodos disponíveis e atribuindo-os a uma variável, conforme abaixo :

```
pct_memoria = psutil.virtual_memory().percent
pct_cpu = psutil.cpu_percent()
pct_disco = psutil.disk_usage('.').percent
```

3.4 - IP significa Internet Protocol. E um número associado a cada aparelho conectado a uma rede para comunicação. Um endereço IP tem duas funções principais: identificação de host ou interface de rede e endereçamento de localização .Usamos também o psutil para coletar dados do IP. Porém a chave de rede varia de acordo com o sistema operacional, por isso devemos checar o sistema operacional para acessarmos a chave correta, conforme abaixo :

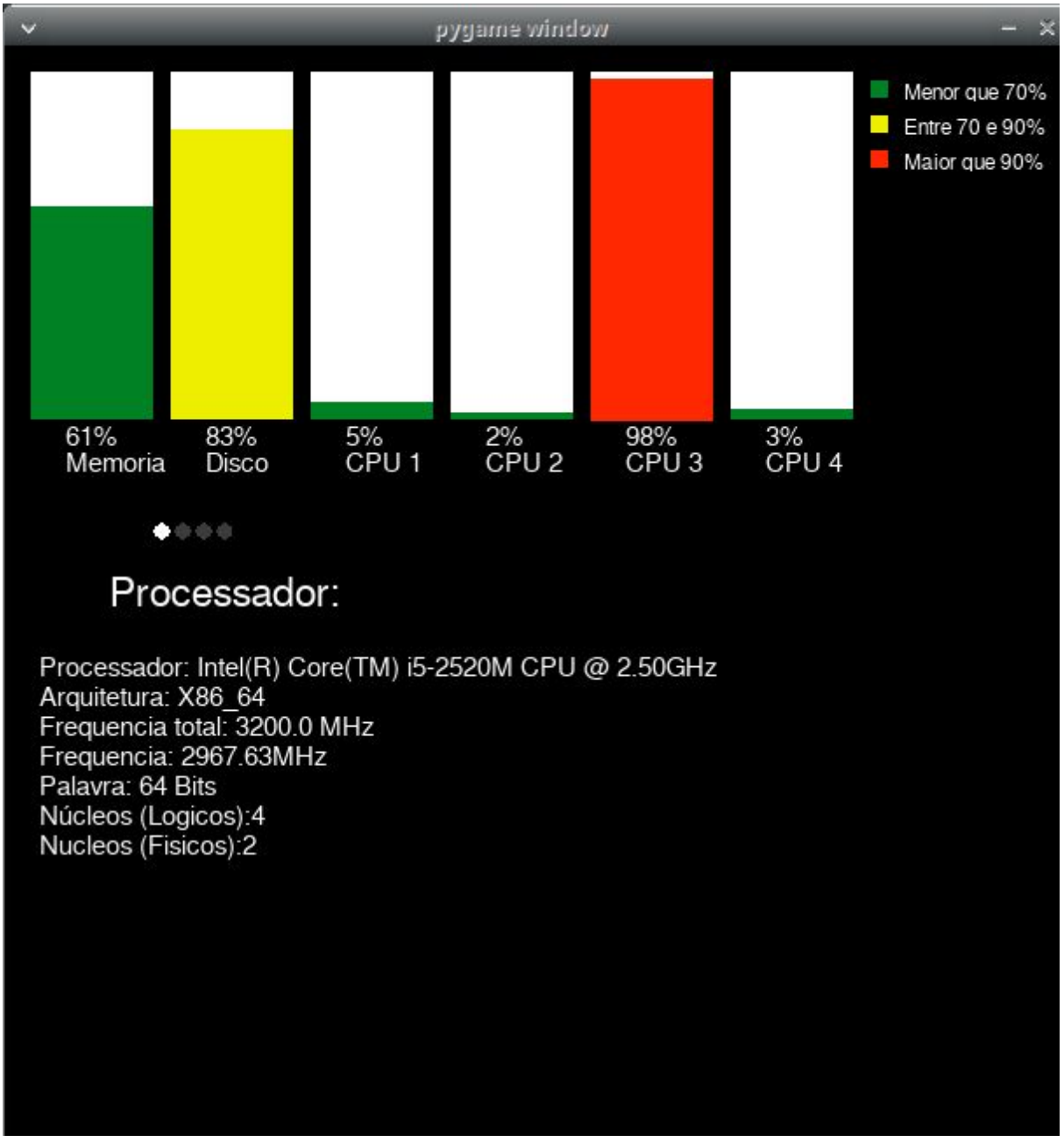
```
rede=''
sistema = platform.system()
if sistema == 'Linux':
    rede='wlp3s0'
elif sistema == 'Windows':
    rede='wlan0'
else:
    rede='eth0'

ip = 'IP: '+ psutil.net_if_addrs()[rede][0].address
```

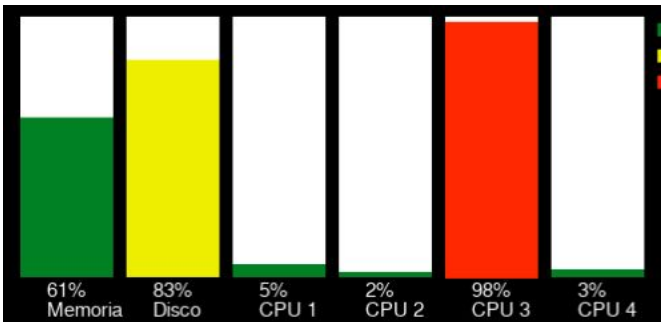
4. APRESENTAÇÃO VERSÃO 3.0

Cumpridas as etapas anteriores, uma nova versão, com agregação de outras funcionalidades, foram acopladas ao Software, a saber :

Conforme telas abaixo :



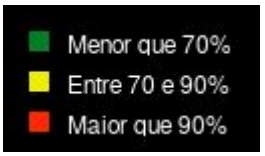
4.1 Barras



As barras de memória e de disco foram colocadas no começo pois elas são fixas. Em seguida as barras de CPU aparecem de acordo com o número de CPUs que o computador atraves da funcao psutil.cpu_count(). Com isso, o número de CPUs foi considerado para fazer a largura da tela. Foram feitas através de uma função que recebe o nome, a porcentagem e o index da barra para que o alinhamento de acordo com o index possa ser feito corretamente. As barras de memória e de Disco são fixas, as de cpus variam de acordo com o computador, foi feito um for nas cpus para desenhar todas.

```
desenhar_barra('Memoria',pct_memoria/100,0)
    desenhar_barra('Disco',pct_disco/100,1)
    for i,cpu in enumerate(psutil.cpu_percent(interval=1,percpu=True)):
        desenhar_barra('CPU '+str(i+1),cpu/100,2+i)
```

4.2 Legenda



Somente textos com quadrados do lado com a respectiva cor, vale lembrar que o tamanho dessa superfície também foi considerada quando fazer o cálculo da largura da tela

```
def desenhar_legenda():

    def desenhar_legenda_unica(cor,label,cont):
        #Seta a font e tamanho
        myfont = pygame.font.SysFont("Arial", 12)
        #Seta a surface
        surface = pygame.surface.Surface((TAMANHO_LEGENDA+10, 12))
        #Seta o texto
        text= myfont.render(label, 1, WHITE)
        #Desenha o quadrado da cor
        pygame.draw.rect(surface,cor,(0,0,10,10))
        #Escreve o texto
        surface.blit(text, (19 , 0))
        #Desenha a surface na tela
        DISPLAY.blit(surface, ( LEGENDA_X, 20+cont*20 ))

    desenhar_legenda_unica(GREEN,'Menor que 70%',0)
    desenhar_legenda_unica(YELLOW,"Entre 70 e 90%",1)
    desenhar_legenda_unica(RED,'Maior que 90%',2)
```

4.3 - Informações do Processador :

As informações acima de 4.1 a 4.13, foram feitas através do módulo psutil e cpuinfo. Cada informação foi colocada em uma lista de acordo com a aba e cada aba foi colocada em uma lista com todas as abas assim podemos chamar a função de desenhar as informações dinamicamente. Algumas das informações variam de acordo com o SO, com isso e criado uma lista principal (informacoes que todos os SOs contém) e uma lista secundária (de acordo com cada SO). E no final elas são concatenadas.

```
#Memoria
mem = psutil.virtual_memory()
memoria_total="Memoria Total:"+ inGB(mem.total)+ " GB"
memoria_disponivel="Memoria Disponivel:"+ inGB(mem.available)+ " GB"
memoria_livre="Memoria Livre:"+ inGB(mem.free)+ " GB"
memoria_usada="Memoria Usada:"+ inGB(mem.used)+ " GB"

net= psutil.net_if_addrs()[rede][array]
ip= 'IP: '+ net.address
netmask='NetMask: '+net.netmask
family='Family: ' + str(net.family)
ptp='Ptp: '+str(net.ptp)
memoria_os=[]
net_os=[]
if sistema == 'Linux':
    memoria_buffers="Buffers:"+ inGB(mem.buffers)+ " GB"
    memoria_cached="Cached:"+ inGB(mem.cached)+ " GB"
    memoria_shared="Compartilhada:"+  inGB(mem.shared)+ " GB"
    memoria_slab="Slab:"+ inGB(mem.slab)+ " GB"
    memoria_ativa="Memoria Ativa:"+ inGB(mem.active)+ " GB"
    memoria_inativa="Memoria Inativa:"+ inGB(mem.inactive)+ " GB"
    memoria_os=[memoria_buffers,memoria_cached,memoria_shared,memoria_slab,memoria_ativa,memoria_inativa]
    net_os=['BroadCast: '+net.broadcast]
elif sistema == 'Windows':
    print("nenhum extra no windows")
elif sistema == 'Darwin':
    memoria_ativa="Memoria Ativa:"+ inGB(mem.active) + " GB"
    memoria_inativa="Memoria Inativa:"+ inGB(mem.inactive) + " GB"
    memoria_wired="Wired: "+inGB(mem.wired)+ " GB"
    memoria_os=[memoria_ativa,memoria_inativa,memoria_wired]
    net_os=['BroadCast: '+net.broadcast]
```

4.4 - Indexação:

A indexação foi feita capturando os cliques da seta esquerda, direita e barra de espaço.Definimos uma variável index que pode ser aumentado ou diminuído seguindo a regra maior que zero para poder diminuir o index e menor que 3 para poder aumentar o index, o index=4, a barra especial. Para acessarmos o resumo salvamos uma cópia do index e colocamos o index=4. Assim podemos usar a cópia do index para retornar para a respectiva aba quando ativamos e desativamos o resumo com a barra de espaço. A bola branca significa que a respectiva aba que está selecionada.As 4 juntas sao o resumo.

```
processador_info=['Processador:',nome,arquitetura,frequencia_total,frequencia,bits,cpuscount,cpuscount
memoria_info=['Memoria: ', memoria_total,memoria_disponivel,memoria_usada,memoria_livre]+memoria_os
disco_info=['Disco:',disco_total,disco_usado,disco_livre]
net_info=['Rede:',ip,netmask,family,ptp]+net_os
resumo_info=['Resumo: ',nome,cpuscount,memoria_total,memoria_disponivel,memoria_usada,memoria_livre,di

infos=[processador_info,memoria_info,disco_info,net_info,resumo_info]
```

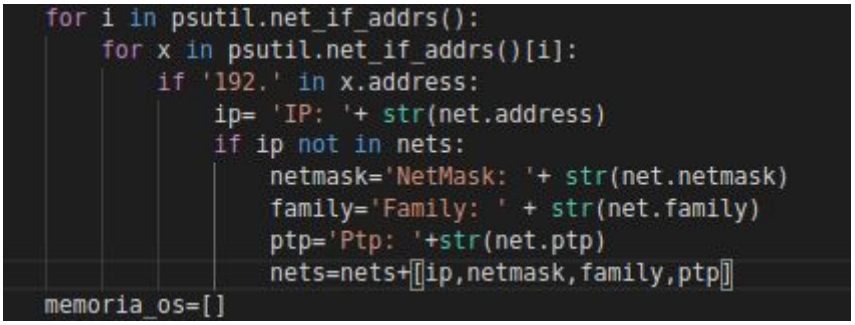
Existem diversos tipos de arquitetura diferentes de CPUs, podemos citar como exemplo as cpus x86 e ARM. A diferença começa na tecnologia empregada para a construção desses dois tipos de processadores, sendo que o x86 utilizam técnicas que permitem uma maior velocidade na execução das tarefas, enquanto por outro lado os processadores baseados em ARM são construídos de acordo com RISC (Reduced Instruction Set Computer), que como o nome já diz, tem como objetivo serem mais simples.

Palavra é a unidade natural de informação utilizada em cada processador, sendo que cada processador trabalha com seu tamanho de palavra fixo e definido, por exemplo processadores de 32 bits e 64 bits, a palavra é usada para realizar a comunicação entre a memória principal e a CPU e também define qual quantidade de bits que serão processador por vez pela CPU.

Núcleo físicos são as partes realmente existentes fisicamente na CPU para a realização dos processos, enquanto os núcleos lógicos são partes virtuais criadas para aumentar a velocidade de processamento das tarefas, os núcleos lógicos

emulam a existência de uma maior quantidade de núcleos do que realmente existem no computador. Os núcleos lógicos são criados para ajudar os núcleos físicos quando esses estão sobrecarregados, eles recebem as tarefas a serem processadas e executam como se fossem núcleos físicos.

Com o intuito de recuperar todos os possíveis IPs, a seguinte lógica foi utilizada:



```
for i in psutil.net_if_addrs():
    for x in psutil.net_if_addrs()[i]:
        if '192.' in x.address:
            ip= 'IP: '+ str(net.address)
            if ip not in nets:
                netmask='NetMask: '+ str(net.netmask)
                family='Family: ' + str(net.family)
                ptp='Ptp: '+str(net.ptp)
                nets=nets+[ip,netmask,family,ptp]
memoria_os=[]
```

5. APRESENTAÇÃO VERSÃO 4.0

Para corrigir um erro no TP3, a função str() foi adicionada a cada variável recuperada das bibliotecas para evitar erro de concatenação.

```
net= psutil.net_if_addrs()[rede][array]
ip= 'IP: '+ str(net.address)
netmask='NetMask: '+ str(net.netmask)
family='Family: ' + str(net.family)
ptp='Ptp: '+str(net.ptp)
```

No TP3, foi possível perceber um delay no programa. Como as barras de CPUs foram desenvolvidas no intuito de serem desenhadas dinamicamente de acordo com a quantidade de CPUs do computador. A função de calcular porcentagem da CPUs era chamada com um intervalo de um segundo dentro do loop principal. Logo,
todo o resto do programa ficava travado esperando esse intervalo. Dessa maneira, quando as setas eram pressionadas, era possível notar um delay.

```
for i,cpu in enumerate(psutil.cpu_percent(interval=1, percpu=True)):
    desenhar_barra('CPU '+str(i+1),cpu/100,2+i)
```

Para corrigir esse programa, uma variável contadora e uma que armazena as porcentagens de CPU foram criadas globalmente:

5.1 Variáveis globais

```
CPUS=psutil.cpu_percent(interval=0.1, percpu=True)
cont=0
```

A variável contadora e implementada toda vez que passa no loop principal, caso o contador seja igual a 90, atualizamos a variável que armazena as porcentagens das CPUs e zeramos o contador para que possa ser incrementado de novo e seja possível que essa atualização de das porcentagens das CPUs sejam atualizadas após 90 ciclos sem comprometer o resto do código.

5.2 Loop principal

```
cont=cont+1
if(cont==90):
    CPUS=psutil.cpu_percent(percpu=True)
    cont=0
```

A partir daqui, foi criado um sistema de navegação clicando as teclas 1,2,3 para mudar entre as abas. Sendo a 1 o monitoramento, 2 navegação em arquivos do computador e 3 visualização dos processos.

```
if aba == 1:
        desenhar_monitoramento()
    if aba == 2:
        desenha_arquivos(PATH)
    if aba == 3:
        desenha_processos()
```

Navegação em arquivos e diretórios:

```
.==================================.==========.====================.====================.
| Voltar  |PATH:/home/rodrigo/ppython/projetobloco
.==================================.==========.====================.====================.
|               Name               |   Size   |      Criado em     |    Modificado em   |
.==================================.==========.====================.====================.
|asdf                              |4.096     |04-10-2020 00:33    |26-09-2020 21:18    |
|tp2.py                            |4.017     |27-09-2020 21:33    |12-09-2020 18:40    |
|.git                              |4.096     |28-09-2020 10:54    |04-10-2020 19:21    |
|tp4.py                            |16.314    |04-10-2020 20:25    |04-10-2020 20:25    |
|tp3.py                            |8.103     |04-10-2020 19:21    |26-09-2020 20:04    |
|Rodrigo_Gomes_TP3_PB.pdf          |273.195   |28-09-2020 20:39    |14-09-2020 01:49    |
.==================================.==========.====================.====================.
 Total ........  : 6 Arquivos e Diretorios
 Total de bytes utilizados : 309.821 Kbytes
.==================================.==========.====================.====================.
```

Para a navegação de arquivos ser possível foi criada uma variável global que controla o diretório que estamos olhando:

```
PATH = os.path.abspath('./')
```

Uma função que altera o path global foi criada para que seja possível voltar ao clicar no botão “voltar”.

```
def voltarPath():
    global PATH
    aux=PATH.split('/')
    if len(aux)>2:
        aux.pop()
        PATH='/'.join(aux)
```

Para entrar no diretório ou abrir o arquivo a função recebe o path, caso o path seja um arquivo, ele chama o comando respectivo ao sistema operacional para abri-lo.

aso seja um diretório, ele sobrescreve a variavel global PATH. Assim e possível com que entramos no diretório.

```
def clickEntrar(namePath):
    global PATH
    if  os.path.isfile(namePath):
        if sistema == 'Linux':
            osCommandString = 'xdg-open '+namePath
        elif sistema == 'Windows':
            osCommandString = namePath
        elif sistema == 'Darwin':
            osCommandString = 'open '+namePath

        os.system(osCommandString)
    else:
        PATH=namePath
```

Para melhorar a usabilidade, foi criada a funcionalidade de passar o mouse em cima do “voltar” e dos arquivos e diretórios e os respectivos botões aumentarem (hover). Esse comportamento e possivel gracas a funcao escrever(), que recebe o index da linha que desenha e verifica se esse índice e igual ao index que o mouse está em cima (variável Global) assim escreve com a fonte maior ou menor.

```
def escrever(label,idx=None):
      if idxMouseOnTop == idx:
          textos.append(myfontbig.render(label,1, WHITE))
      else:
          textos.append(myfont.render(label,1, WHITE))
```

O desenho da tela foi feita atraves da tecnica .format() dado em sala de aula e aplicando ao pygame. Usando a biblioteca os para conseguir as informações dos arquivos e diretórios. Como tamanho, datas de criação e modificação.

```
escrever(t1.format("","","",""))
escrever(t0.format('','PATH:'+PATH))
escrever(t1.format("","","",""))
escrever(t2.format("Name","Size","Criado em","Modificado em"))
escrever(t1.format("","","",""))

totArquivos = 0      You, 19 hours ago • tp4 first
totBytes    = 0

dic = {}
for idx,name in enumerate(dirs):
    namePath=PATH+'/'+name
    dic[namePath] = []
    dic[namePath].append(os.stat(namePath).st_size)
    dic[namePath].append(os.stat(namePath).st_atime)
    dic[namePath].append(os.stat(namePath).st_mtime)

    dtatime = datetime.fromtimestamp(dic[namePath][1])
    formatdtatime = ("{:{dfmt} {tfmt}}".format(dtatime, dfmt="%d-%m-%Y", tfmt="%H:%M"))
    dtmtime = datetime.fromtimestamp(dic[namePath][2])
    formatdtmtime = ("{:{dfmt} {tfmt}}".format(dtmtime, dfmt="%d-%m-%Y", tfmt="%H:%M"))
    t="|{:35}|{:10}|{:21}|{:21}|"

    escrever(t.format(name,str(dic[namePath][0]/1000),formatdtatime,formatdtmtime),idx)
    totArquivos +=1
    totBytes = totBytes + dic[namePath][0]

escrever(t1.format("","","",""))

escrever(" Total ........  : "+str(totArquivos)+" Arquivos e Diretorios")
escrever(" Total de bytes utilizados : "+str(totBytes/1000)+" Kbytes")
escrever(t1.format("","","",""))
```

Para melhorar a usabilidade e a parte gráfica, a função escreve() foi modificada para que os diretórios sejam exibidos na cor amarela:

```
def escrever(label,namePath='',idx=None):
    COLOR=WHITE
    if os.path.isdir(namePath):
        COLOR=YELLOW
    if idxMouseOnTop == idx:
        textos.append(myfontbig.render(label,1, COLOR))
    else:
        textos.append(myfont.render(label,1, COLOR))
```

OS X PSUTIL

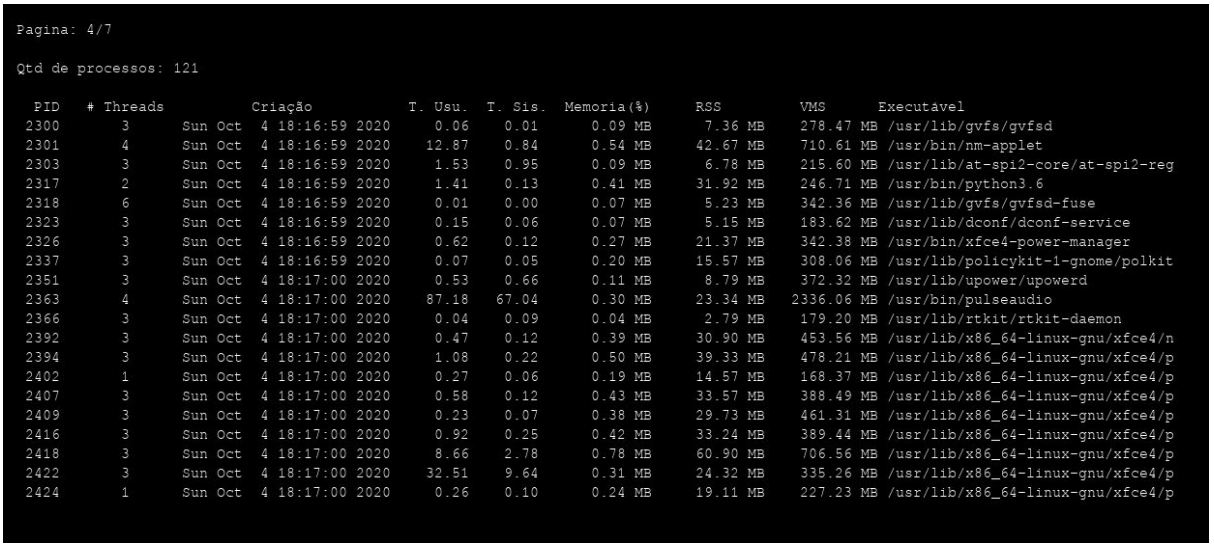
Este módulo fornece uma maneira simples de usar funcionalidades que são dependentes de sistema operacional. Se desejar ler ou escrever um arquivo, veja open(); se o que quer é manipular estruturas de diretórios, veja o módulo os.path; e se quiser ler todas as linhas, de todos os arquivos, na linha de comando, veja o módulo fileinput. Para criar arquivos e diretórios temporários, veja o módulo tempfile; e, para manipulação em alto nível de arquivos e diretórios, veja o módulo shutil.

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

O modulo OS e mais usado em manipulacao de arquivos, diretorios, ja o Psutil é utilizado para obtencao dos recursos do hardware (memoria, disco, CPU …)

6. APRESENTAÇÃO VERSÃO 5.0

6.1 Processos



```
Pagina: 4/7

Qtd de processos: 121

  PID   # Threads          Criação          T. Usu.  T. Sis.  Memoria(%)     RSS         VMS      Executavel
 2300       3      Sun Oct  4 18:16:59 2020     0.06     0.01      0.09 MB      7.36 MB    278.47 MB /usr/lib/gvfs/gvfsd
 2301       4      Sun Oct  4 18:16:59 2020    12.87     0.84      0.54 MB     42.67 MB    710.61 MB /usr/bin/nm-applet
 2303       3      Sun Oct  4 18:16:59 2020     1.53     0.95      0.09 MB      6.78 MB    215.60 MB /usr/lib/at-spi2-core/at-spi2-reg
 2317       2      Sun Oct  4 18:16:59 2020     1.41     0.13      0.41 MB     31.92 MB    246.71 MB /usr/bin/python3.6
 2318       6      Sun Oct  4 18:16:59 2020     0.01     0.00      0.07 MB      5.23 MB    342.36 MB /usr/lib/gvfs/gvfsd-fuse
 2323       3      Sun Oct  4 18:16:59 2020     0.15     0.06      0.07 MB      5.15 MB    183.62 MB /usr/lib/dconf/dconf-service
 2326       3      Sun Oct  4 18:16:59 2020     0.62     0.12      0.27 MB     21.37 MB    342.38 MB /usr/bin/xfce4-power-manager
 2337       3      Sun Oct  4 18:16:59 2020     0.07     0.05      0.20 MB     15.57 MB    308.06 MB /usr/lib/policykit-1-gnome/polkit
 2351       3      Sun Oct  4 18:17:00 2020     0.53     0.66      0.11 MB      8.79 MB    372.32 MB /usr/lib/upower/upowerd
 2363       4      Sun Oct  4 18:17:00 2020    87.18    67.04      0.30 MB     23.34 MB   2336.06 MB /usr/bin/pulseaudio
 2366       3      Sun Oct  4 18:17:00 2020     0.04     0.09      0.04 MB      2.79 MB    179.20 MB /usr/lib/rtkit/rtkit-daemon
 2392       3      Sun Oct  4 18:17:00 2020     0.47     0.12      0.39 MB     30.90 MB    453.56 MB /usr/lib/x86_64-linux-gnu/xfce4/n
 2394       3      Sun Oct  4 18:17:00 2020     1.08     0.22      0.50 MB     39.33 MB    478.21 MB /usr/lib/x86_64-linux-gnu/xfce4/p
 2402       1      Sun Oct  4 18:17:00 2020     0.27     0.06      0.19 MB     14.57 MB    168.37 MB /usr/lib/x86_64-linux-gnu/xfce4/p
 2407       3      Sun Oct  4 18:17:00 2020     0.58     0.12      0.43 MB     33.57 MB    388.49 MB /usr/lib/x86_64-linux-gnu/xfce4/p
 2409       3      Sun Oct  4 18:17:00 2020     0.23     0.07      0.38 MB     29.73 MB    461.31 MB /usr/lib/x86_64-linux-gnu/xfce4/p
 2416       3      Sun Oct  4 18:17:00 2020     0.92     0.25      0.42 MB     33.24 MB    389.44 MB /usr/lib/x86_64-linux-gnu/xfce4/p
 2418       3      Sun Oct  4 18:17:00 2020     8.66     2.78      0.78 MB     60.90 MB    706.56 MB /usr/lib/x86_64-linux-gnu/xfce4/p
 2422       3      Sun Oct  4 18:17:00 2020    32.51     9.64      0.31 MB     24.32 MB    335.26 MB /usr/lib/x86_64-linux-gnu/xfce4/p
 2424       1      Sun Oct  4 18:17:00 2020     0.26     0.10      0.24 MB     19.11 MB    227.23 MB /usr/lib/x86_64-linux-gnu/xfce4/p
```

O módulo psutil foi usado para recuperar os pids dos processos existentes através da função psutil.pids(). Essa função retorna uma lista com os ids dos processos.

Para a paginação, a variável pages é uma lista que armazena listas com seus respectivos processos. A variável “page_size” determina o tamanho da página.

```
pages=[]
        for pid in lista:
            info_process=pegar_info(pid)
            if info_process != None:
                if aux==0 and auxidx==0:
                    pages.append([])

                if auxidx < page_size:
                    pages[aux].append(info_process)
                else:
                    auxidx=0
                    aux=aux+1
                    pages.append([])
                    pages[aux].append(info_process)
                auxidx=auxidx+1
```

Para que a paginação não seja feita desnecessariamente, a quantidade atual de processos é salva e comparada com a nova quantidade , se forem diferente significa que devemos construir a paginação novamente.

```
    qtd_processos=0
    if qtd_processos != len(lista):
        qtd_processos=len(lista)
```

A função pegar_info() recebe o pid e concatena uma string para ser exibida com as informações úteis do processo.

```
def pegar_info(pid):
    try:
        p = psutil.Process(pid)
        texto = '{:^7}'.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()) + " MB"
        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()
        return texto
    except:
        pass
```

A variável page foi criada para que controle a visualização das páginas. Ela é alterada se a página atual for maior que 0 ou menor que o número de páginas

```
        if aba ==3:
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RIGHT and page < len(pages)-1:
                    page=page+1
                if event.key == pygame.K_LEFT and page > 0:
                    page=page-1
```

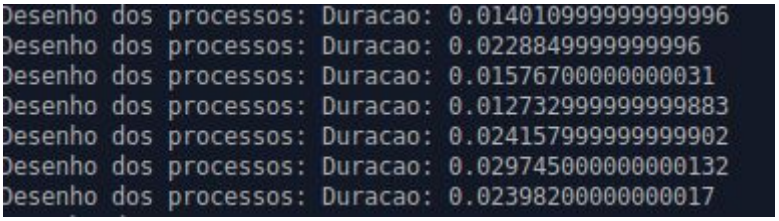
Com isso, basta usar essa variável “page” para acessarmos o index na lista de páginas desejado e imprimir todos os processos chamando a função escrever()

```
  for idx,process in enumerate(pages[page]):
        escrever(process,idx+1)
```

Usando o módulo “time” foi calculado quanto tempo (em clocks) a função de desenhar processo dura para ser executada

```
def desenha_processos():
    initial_time=time.clock()
    .
    . #### Resto da funcao
    .
    end_time=time.clock()
    print('Desenho dos processos: Duracao:',end_time-initial_time)
```

6.2 Resultado



Caso a função time.sleep(2) for chamada, o processo fará uma pausa por 2 segundos para depois prosseguir, abaixo está a mesma função com sleep de 2 segundos.