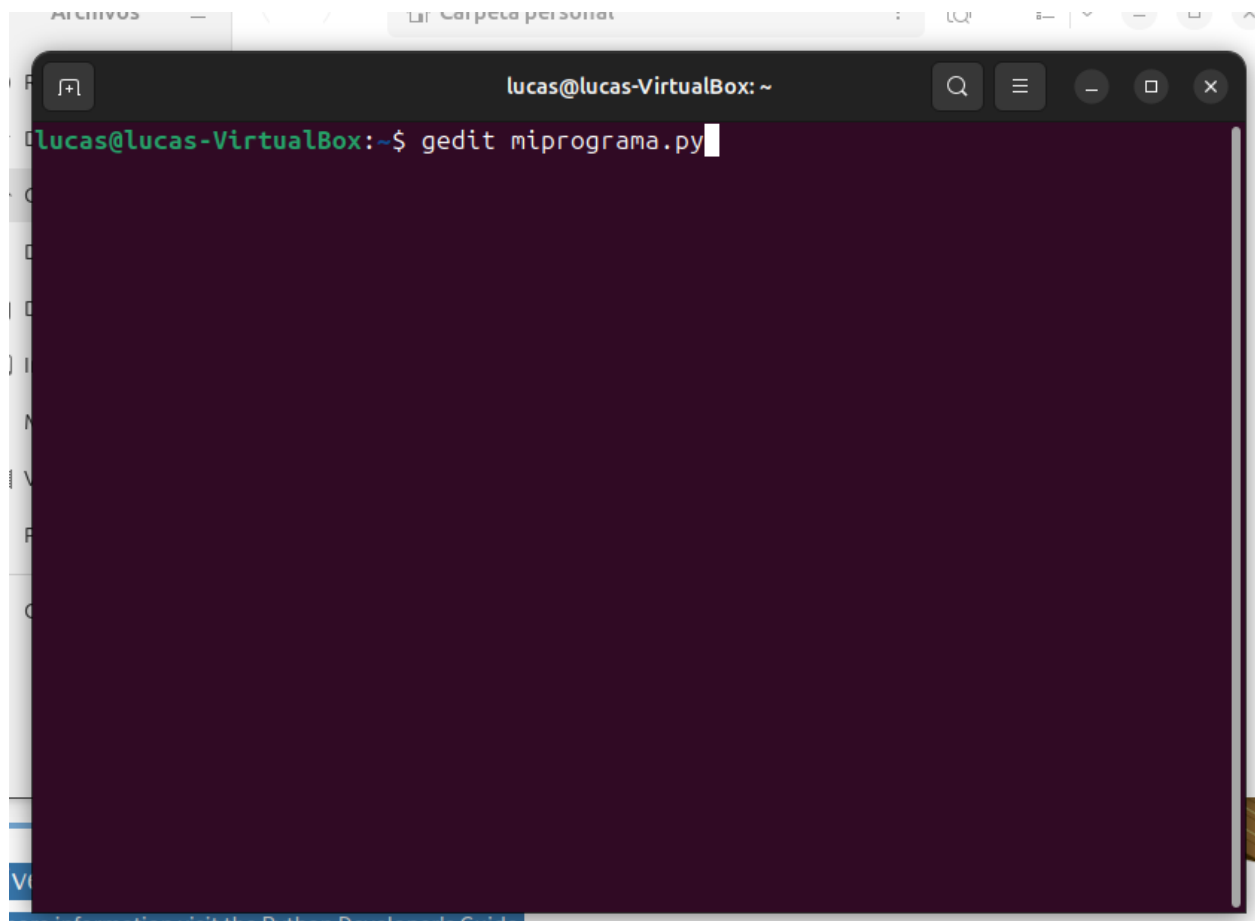


Laboratorio 1: Gestión de Procesos

Para realizar este laboratorio primeramente tuve que instalar Ubuntu en una máquina virtual con las especificaciones necesarias de hardware para poder realizar las actividades. Posteriormente tuve que instalar las aplicaciones necesarias, tales como: htop, python3 y gedit. Una vez con eso pude avanzar con el desarrollo de las actividades.

Primeramente codifiqué un programa sencillo en lenguaje Python utilizando gedit que sería básicamente un bloc de notas de Linux, adjunto el programa:



Abro el archivo por medio de la terminal

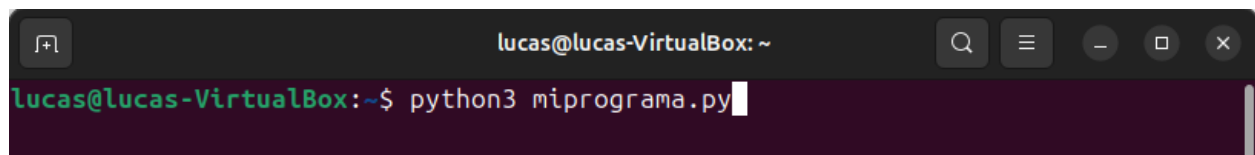


```
1 import time
2 import os
3
4 nombre = input ("Ingrese su nombre: ")
5 print ("Hola ", nombre, " este es un simulador de estados hecho por Lucas Velazquez
   favor aguarde")
6
7 print ("")
8 print ("Iniciando...")
9 time.sleep(2)
10
11 print ("Ejecutando")
12 start = time.time()
13
14 for i in range(10**9):
15     pass
16
17 end = time.time()
18 print ("Terminado")
19
20 print (f"El tiempo del estado 'Ejecutando' a 'Terminado' fue de: {end - start : .2f}
   segundos")
21
```

Python ▾ Anchura del tabulador: 8 ▾ Ln 14, Col 21 INS

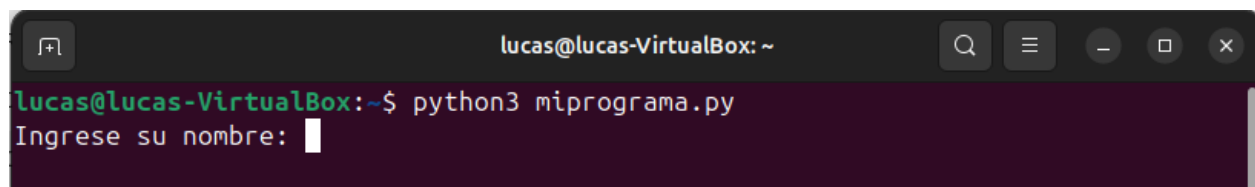
Se abre el archivo gedit con las líneas de código en lenguaje Python

Posteriormente lo podemos ejecutar desde la terminal con el siguiente comando:



```
lucas@lucas-VirtualBox: ~$ python3 miprograma.py
```

Y el programa se ejecutará dentro de la terminal, de la siguiente manera:



```
lucas@lucas-VirtualBox: ~$ python3 miprograma.py
Ingrese su nombre: 
```

En este momento el proceso de Python ya se está ejecutando en modo “Sleeping” ya que está esperando en este caso que se ingrese el “Nombre”. Eso lo podemos ver en htop, que es como el administrador de tareas de Linux. De la siguiente manera podemos abrir htop (anteriormente descargado por medio de los códigos “sudo apt update” y “sudo apt install htop”).

```
lucas@lucas-VirtualBox:~$ htop
```

Y se ejecutará htop, que se ve de la siguiente manera:

```

0[          0.0%] Tasks: 122, 550 thr, 100 kthr; 1 running
1[          0.0%] Load average: 0.04 0.06 0.10
2[          0.7%] Uptime: 01:13:55
Mem[|||||1.18G/2.92G]
Swp[|||||371M/2.92G]

Main I/O
PIDUSER      PRI NI  VIRT  RES  SHR S  CPU% MEM%  TIME+ Command
27465 lucas    20  0 20552 5560 3640 R  1.3  0.2  0:00.29 htop
27459 lucas    20  0 3095M 260M 93400 S  0.0  8.7  0:00.00 /snap/firefox/5751/usr/lib/firefox/firefox
27419 lucas    20  0 19692 5056 3776 S  0.0  0.2  0:00.00 bash
27400 lucas    20  0 29120 9536 6592 S  0.0  0.3  0:00.00 python3 miprograma.py
27380 lucas    20  0 19692 5048 3768 S  0.0  0.2  0:00.00 bash
27379 lucas    20  0 558M 61220 48052 S  0.0  2.0  0:00.00 /usr/libexec/gnome-terminal-server
27378 lucas    20  0 558M 61220 48052 S  0.0  2.0  0:00.00 /usr/libexec/gnome-terminal-server
27377 lucas    20  0 558M 61220 48052 S  0.0  2.0  0:00.00 /usr/libexec/gnome-terminal-server
27375 lucas    20  0 558M 61220 48052 S  0.0  2.0  0:00.00 /usr/libexec/gnome-terminal-server
27374 lucas    20  0 558M 61220 48052 S  0.0  2.0  0:00.00 /usr/libexec/gnome-terminal-server
27373 lucas    20  0 558M 61220 48052 S  0.0  2.0  0:01.50 /usr/libexec/gnome-terminal-server
27008 lucas    20  0 4499M 302M 112M S  0.0 10.1  0:00.00 /usr/bin/gnome-shell
26970 lucas    39 19 2143M 251M 114M S  0.0  8.4  0:00.00 /usr/bin/nautilus --gapplication-service
26969 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.00 /usr/bin/nautilus --gapplication-service
26968 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.00 /usr/bin/nautilus --gapplication-service
26967 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.00 /usr/bin/nautilus --gapplication-service
26966 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.50 /usr/bin/nautilus --gapplication-service
26965 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.46 /usr/bin/nautilus --gapplication-service
26964 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.46 /usr/bin/nautilus --gapplication-service
26956 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.00 /usr/bin/nautilus --gapplication-service
26955 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.01 /usr/bin/nautilus --gapplication-service
26948 lucas    20  0 2143M 251M 114M S  0.0  8.4  0:00.00 /usr/bin/nautilus --gapplication-service
F1Help F2Setup F3SearchF4FilterF5Tree F6SortByF7Nice F8Nice F9Kill F10Quit

```

Podemos apreciar que Python se está ejecutando con el PID 27400 y en la columna de “S” que significa Status o Estado en español, se ve la letra “S” que quiere decir “Sleeping” por lo que no se está ejecutando ningún proceso como mencionamos anteriormente, más bien esta en un estado de espera. Pero al ingresar el nombre, permitiendo que el programa se siga ejecutando el estado del proceso cambia

```
lucas@lucas-VirtualBox:~$ python3 miprograma.py
Ingrese su nombre: Lucas
```

Ingreso el nombre

```
lucas@lucas-VirtualBox: ~$ python3 miprograma.py
Ingrese su nombre: Lucas
Hola Lucas este es un simulador de estados hecho por Lucas Velazquez favor aguarde

Iniciando...
Ejecutando
```

Se ejecutan las líneas de código posteriores al ingreso del nombre y podemos visualizar el cambio de estado en htop

Swp[] 371M/2.92G											
Main I/O											
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
27481	lucas	20	0	29120	9500	6556	R	139.1	0.3	0:05.31	python3 miprograma.py
27465	lucas	20	0	20552	5716	3796	R	2.4	0.2	0:10.77	htop
27419	lucas	20	0	19692	5056	3776	S	0.0	0.2	0:00.00	bash

Podemos ver que el estado cambio de “S” a “R” que quiere decir Running o Corriendo en español, ya que en el momento se estaban ejecutando procesos dentro del programa. Gracias a las siguientes líneas de código podemos ver cuanto tardo el programa en pasar de “Ejecutando” a “Terminado”.

```
start = time.time()
```

```
end = time.time()
```

```
print ("f{end - start : .2f}")
```

Viéndose de la siguiente manera

```
Iniciando...
Ejecutando
Terminado
El tiempo del estado 'Ejecutando' a 'Terminado' fue de: 35
.17 segundos
lucas@lucas-VirtualBox: ~$
```

También podemos forzar a que el programa pase al estado “T”(Stopped) por medio de un comando en la terminal utilizando la PID del proceso que queremos poner en ese estado

```
lucas@lucas-VirtualBox:~$ kill -SIGSTOP 4392
```

Ejecutamos con el PID del proceso de Python, dejándolo de la siguiente manera

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
4411	lucas	20	0	19692	5056	3776	S	0.0	0.2	0:00.00	bash
4392	lucas	20	0	29120	9476	6532	T	0.0	0.3	0:00.00	python3 m
4220	lucas	20	0	20060	4992	3584	R	2.0	0.2	0:09.70	htop

En la columna de estado cambió a “T”(stopped). Para volver a reanudar ejecutamos el siguiente código

```
lucas@lucas-VirtualBox:~$ kill -SIGCONT 4392
```

Luego para realizar el Scheduling del Sistema Operativo hay que saber primeramente que Ubuntu no utiliza los algoritmos teóricos FIFO(por orden de llegada de procesos) o RoundRobin(% de CPU para cada proceso por cierto periodo de tiempo), sino que utiliza el algoritmo CFS(Completely Fair Scheduler o Planificador completamente justo), que reparte el CPU entre todos los procesos de manera justa, en este caso ejecuté 2 programas que simplemente consumen CPU con una línea de código bastante simple pero se trata de un bucle infinito, es el siguiente:

```
1 while True:
2     x=123456*654321S
```

Como mi Notebook no tiene mucha potencia y claramente mi máquina virtual tampoco, al ejecutar solo una vez el programa ya causaba que el sistema empiece a colapsar, por lo que utilice cpulimit para limitar el uso de CPU con el programa. De la siguiente manera:

```
lucas@lucas-VirtualBox:~$ cpulimit -l 5 -- python3 cpuconsume.py &
```

De esta manera ya pude ejecutar dos veces en segundo plano el programa y podemos ver como el sistema operativo realiza ese reparto de CPU entre los dos procesos que más consumen, para que ambos trabajen de la misma manera, como se puede observar en la imagen. (PID 3824 y 3788)

3824	lucas	20	0	28988	9196	6380	T	94.9	0.3	1:37.56	python3 cpuconsume.py
3810	lucas	20	0	19696	5104	3824	S	0.0	0.2	0:00.00	bash
3789	lucas	20	0	2696	1136	1136	S	0.6	0.0	0:00.40	cpulimit -l 5 -- python3 cpuconsume.py
3788	lucas	20	0	28988	9208	6392	T	95.4	0.3	2:58.86	python3 cpuconsume.py
3767	lucas	20	0	20108	5236	3700	R	1.7	0.2	0:05.00	htop
3753	lucas	20	0	19696	5032	3752	S	0.0	0.2	0:00.00	bash
3715	lucas	20	0	19956	5296	3760	S	0.0	0.2	0:00.03	bash

Por último relicé un deadlock en Python usando el siguiente programa.

```
1 import threading
2 import time
3
4 # creamos dos recursos
5 lock_1 = threading.Lock()
6 lock_2 = threading.Lock()
7
8 def thread1():
9     print ("Hilo 1 Bloqueando recurso 1")
10    lock_1.acquire()
11    time.sleep(1)
12    print ("Hilo 1 Tratando de bloquear recurso 2")
13    lock_2.acquire()
14    print ("Hilo 1 Consiguió ambos recursos")
15    lock_2.release()
16    lock_1.release()
17
18 def thread2():
19     print ("Hilo 2 Bloquando recurso 2")
20    lock_2.acquire()
21    time.sleep(1)
22    print ("Hilo 2 Tratando de Bloquear recurso 1")
23    lock_1.acquire()
24    print ("Hilo 2 Consiguió ambos recursos")
25    lock_1.release()
26    lock_2.release()
27
28 t1 = threading.Thread(target=thread1)
29 t2 = threading.Thread(target=thread2)
30
31 t1.start()
32 t2.start()
33
34 t1.join()
35 t2.join()
```

¿Por qué ocurre el deadlock?

Básicamente porque el hilo 1 o thread1 bloquea el recurso lock_1 y al mismo tiempo el hilo 2 bloquea el recurso lock_2 por lo que cuando los dos hilos quieren acceder al siguiente recurso no pueden hacerlo debido a que ya fueron bloqueados. Viendose de la siguiente manera.

```
lucas@lucas-VirtualBox: ~  
lucas@lucas-VirtualBox:~$ gedit deadlocklucas.py  
lucas@lucas-VirtualBox:~$ python3 deadlocklucas.py  
Traceback (most recent call last):  
  File "/home/lucas/deadlocklucas.py", line 5, in <module>  
    lock_1 = lock.Threading()  
            ^^^^^  
NameError: name 'lock' is not defined  
lucas@lucas-VirtualBox:~$ gedit deadlocklucas.py  
lucas@lucas-VirtualBox:~$ python3 deadlocklucas.py  
Hilo 1 Bloqueando recurso 1  
Hilo 2 Bloqueando recurso 2  
Hilo 1 Tratando de bloquear recurso 2  
Hilo 2 Tratando de Bloquear recurso 1
```

El programa se queda congelado en ese punto, para solucionar el deadlock es bastante simple, solo hay que hacer que los dos hilos accedan a los recursos en el mismo orden.

```
lucas@lucas-VirtualBox: ~  
lucas@lucas-VirtualBox:~$ python3 deadlocklucas.py  
Hilo 1 Bloqueando recurso 1  
Hilo 2 Bloqueando recurso 1  
Hilo 1 Tratando de bloquear recurso 2  
Hilo 1 Consiguió ambos recursos  
Hilo 2 Tratando de Bloquear recurso 2  
Hilo 2 Consiguió ambos recursos
```

Para lograr esto obviamente hay que modificar el código de la siguiente manera para que ambos threads tengan el mismo orden:

```
8 def thread1():
9     print ("Hilo 1 Bloqueando recurso 1")
10    lock_1.acquire()
11    time.sleep(1)
12    print ("Hilo 1 Tratando de bloquear recurso 2")
13    lock_2.acquire()
14    print ("Hilo 1 Consiguió ambos recursos")
15    lock_2.release()
16    lock_1.release()
17
18 def thread2():
19    print ("Hilo 2 Bloquando recurso 1")
20    lock_1.acquire()
21    time.sleep(1)
22    print ("Hilo 2 Tratando de Bloquear recurso 2")
23    lock_2.acquire()
24    print ("Hilo 2 Consiguió ambos recursos")
25    lock_2.release()
26    lock_1.release()
27
```