

Estatus



Nombre del alumno: José Ulises Vallejo Sierra

Código: 219747905

Sección: D06

Curso: Computación Tolerante a Fallas

Nombre del profesor: Michel Emanuel López Franco

Realizar un programa que sea capaz de revisar el estatus de tu aplicación.

Técnica a utilizar: psutil

En esta nueva etapa del curso, se nos pone como reto generar un servicio o en este caso aplicación externa a la aplicación principal, que revise constantemente el estatus de ésta última, de manera que detecte si el programa se está ejecutando normal y correctamente o si es que hubo algún cierre inesperado de la aplicación.

Para mi caso, reutilicé la aplicación de la entrega pasada, la cual consistía en el llenado de datos de una bitácora de animales para un zoológico, y la cual es ya capaz de recuperar los datos agregados tras un cierre inesperado, pero para esta ocasión y para mejorar o enriquecer más la parte tolerante a los fallos, hemos planteado una solución que permite detectar si esta aplicación está en uso o si es que hubo algún tipo de error que haya finalizado la ejecución inesperadamente, de ser así se notificará e inmediatamente se dará pauta para por medio de este script con el servicio o módulo psutil mostrado por el docente, levantar la aplicación de nuevo, permitiendo así ser más tolerante contra fallas inesperadas.

A continuación, se presenta el programa realizado acerca de los registros de las especies de un zoológico, así como la simulación o ejemplificación de como nuestro script es capaz de levantar y poner a trabajar de nuevo nuestra aplicación

Programa: Captura de animales en un Zoológico.

```
60 while True:
61     limpiar()
62     print("----- MENÚ BITACORA -----")
63     print("1)-Restaurar bitacora (cargar de agenda.txt)")
64     print("2)-Agregar Animal")
65     print("3)-Ver Bitacora")
66     print("0) Salir")
67     opcion = input("Eliga una opcion: ")
68
69     if opcion == "1":
70         miBitacora.restaurar_bitacora()
71         pausar()
72
73     elif opcion == "2":
74         limpiar()
75         especie = input("Especie del animal: ")
76         sexo = input("Sexo del animal: ")
77         raza = input("Raza del animal: ")
78         peso = input("Peso del animal: ")
79         tipo = input("Aéreos / Acuáticos / Terrestres: ")
80         miBitacora.añadir_animal(Animal(especie, sexo, raza, peso, tipo))
81         pausar()
82
83     elif opcion == "3":
84         limpiar()
85         miBitacora.ver_bitacora()
86         pausar()
87
88     elif opcion == "0":
89         break
```

Figura 1. Uso de un menú que ayude al usuario a la utilización de la aplicación y brinde una interfaz en consola para facilitar esto.

```
1  import pickle, os
2
3  # Funcion para limpiar la consola
4  def limpiar():
5      os.system("cls")
6
7  # Funcion para pausar la consola
8  def pausar():
9      os.system("PAUSE")
```

Figura 2. Importación de las librerías pickle (módulo para la serialización de los datos) y creación de funciones para limpiar y pausar pantalla

Creamos la clase animal, la cual tiene todas las propiedades a especificar de éste en su constructor, y una función que ayude la visualización de los datos

```
11 class Animal:
12     def __init__(self, especie, sexo, raza, peso, tipo):
13         self.especie = especie
14         self.sexo = sexo
15         self.raza = raza
16         self.peso = peso
17         self.tipo = tipo
18         print('Se ha añadido un:',self.especie)
19
20     def __str__(self):
21         return 'Especie: {} Sexo: {} Raza: {} Peso: {} Tipo: {}'.format(self.especie, self.sexo,
22         self.raza, self.peso, self.tipo)
```

Figura 3. Creación de la clase animal

Ahora, creamos la clase bitácora, la cual contará con todas las funciones para el manejo de los datos de los registros de la clase animal, primero creamos una lista de animales donde alojaremos cada animal que se irá creando, luego con la función añadir, y el parámetro donde recibirá al animal, insertamos dicho animal a la lista e invocamos el método guardar, a fin de que se guarde cada dato ingresado

```
24 class Bitacora:
25
26     animales = []
27
28     def añadir_animal(self, p):
29         self.animales.append(p)
30         self.guardar()
31
```

Figura 4. Creación de la clase bitácora y función para añadir un animal

Ahora creamos la función para restaurar la bitácora del archivo de texto binario, creamos el fichero y lo declaramos como ab+ para escribir sin sobre escribir texto binario, y posicionamos el fichero al inicio, si no se había guardado nada, imprimimos mensaje de fichero vacío, de lo contrario, hacemos el volcado del fichero a la lista con pickle.load() y por último mandamos el mensaje de cuantos animales se restauraron con la cantidad de elementos que tenga la lista.

```
33     def restaurar_bitacora(self):
34         fichero = open('bitacora', 'ab+')
35         fichero.seek(0)
36         try:
37             self.animales = pickle.load(fichero)
38         except:
39             print("El fichero está vacío")
40         finally:
41             fichero.close()
42             print("Se han cargado {} animales".format(len(self.animales)))
43
```

Figura 5. Función para restaurar los datos (volcado de datos del fichero binario)

Ahora con una función para mostrar los datos almacenados en bitácora (lista animales) mediante un for imprimimos cada elemento y esté aparecerá con el formato establecido en la clase Animal para, si la lista está vacía, se imprime dicho mensaje

```
45     def ver_bitacora(self):
46         if len(self.animales) == 0:
47             print("La bitácora está vacía")
48             return
49         for p in self.animales:
50             print(p)
```

Figura 6. Función para visualizar los datos de la bitácora

Por último, creamos la función para guardar los datos ingresados (esta se llama automáticamente después de cada animal añadido) la cual hace la carga o escritura binaria (wb) de los datos de la lista de animales al fichero con pickle.dump()

```
53     def guardar(self):
54         fichero = open('bitacora', 'wb')
55         pickle.dump(self.animales, fichero)
56         fichero.close()
```

Script status.py que revisa el estado del programa, notifica y levanta el servicio.

Importamos la librería psutil que permite detectar y gestionar la ejecución de un programa o supervisar el mismo, además de las librerías os y time que utilizaremos más adelante para controlar el programa.

```
status.py > ...  
1  import psutil  
2  import os, time  
3  from colorama import Fore, Style
```

Figura 8. Importación de librerías

Mediante un ciclo infinito revisamos en tiempo real la ejecución del programa, donde con ayuda de una bandera y un for para cada proceso en ejecución del tipo main para Python, detectamos dicho proceso levantando la bandera

```
5  while True:  
6      procesos = 0  
7      for proc in psutil.process_iter():  
8            
9          if proc.name().lower() == 'main.exe' or proc.name().lower() == 'python.exe':  
10             procesos += 1
```

Si la bandera se levantó, quiere decir que la app no está en ejecución, por lo que en este caso se pregunta si quiere ser restaurada, si elige una opción positiva, la aplicación principal se levanta; Si la bandera no se levanta, entonces la app ya está en ejecución y se manda un mensaje con dicho estado.

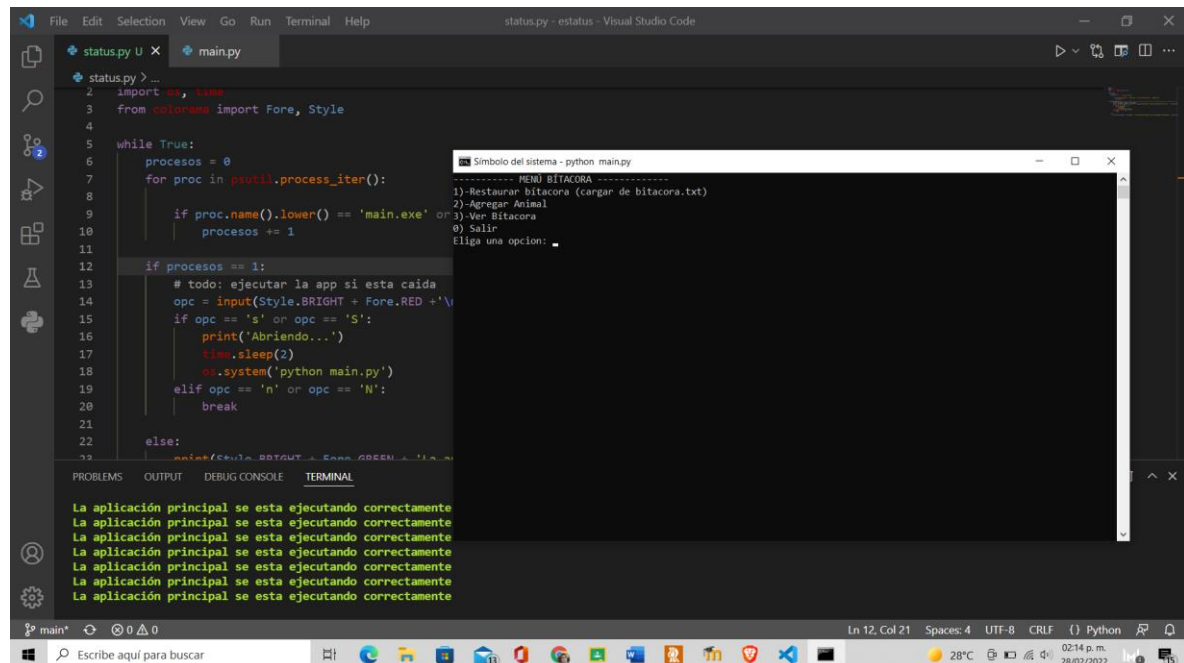
```
12  if procesos == 1:  
13      # todo: ejecutar la app si esta caida  
14      opc = input(Style.BRIGHT + Fore.RED + '\nLa aplicación está cerrada, desea abrirla? (s/n): ' + Style.RESET_ALL)  
15      if opc == 's' or opc == 'S':  
16          print('Abriendo...')  
17          time.sleep(2)  
18          os.system('python main.py')  
19      elif opc == 'n' or opc == 'N':  
20          break  
21    
22  else:  
23      print(Style.BRIGHT + Fore.GREEN + 'La aplicación principal se esta ejecutando correctamente' + Style.RESET_ALL)
```

Ejecución del programa

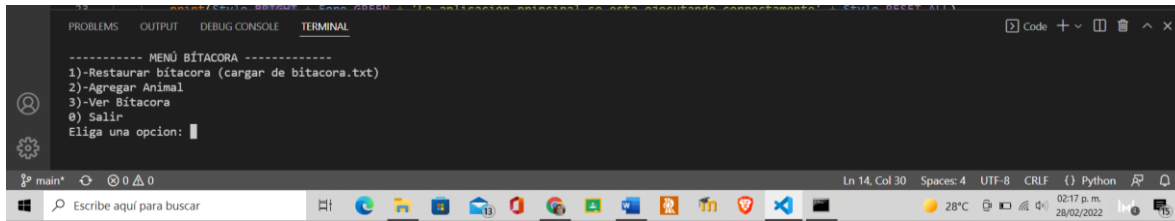
```
Símbolo del sistema - python main.py

----- MENÚ BITACORA -----
1)-Restaurar bitacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bitacora
0) Salir
Eliga una opcion: _
```

Ejecutamos la aplicación en la terminal y al estar corriendo revisamos el estatus

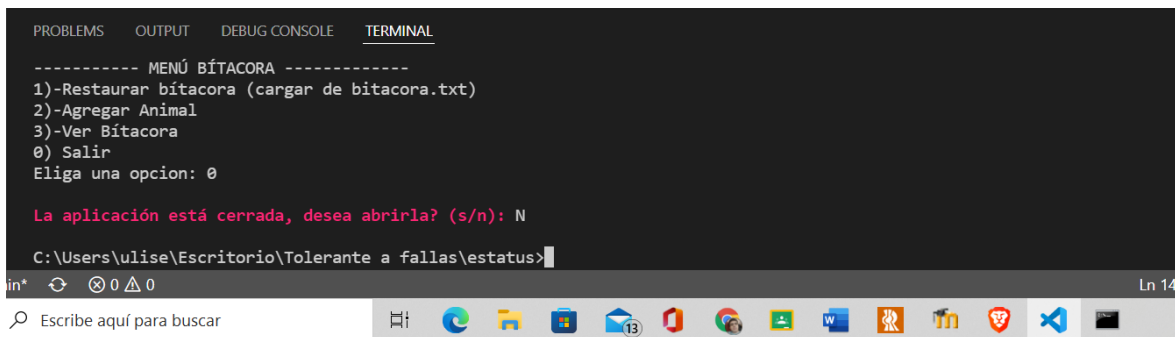


Ejecutamos `status.py` y al estar corriendo nos informa en la terminal que la aplicación se ejecuta correctamente



```
----- MENÚ BITACORA -----
1)-Restaurar bitacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bitacora
0) Salir
Eliga una opcion: 
```

Observamos que el programa se recuperó con éxito, por lo que se cumple el objetivo de esta asignación 😊.



```
----- MENÚ BITACORA -----
1)-Restaurar bitacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bitacora
0) Salir
Eliga una opcion: 0

La aplicación está cerrada, desea abrirla? (s/n): N

C:\Users\ulise\Escritorio\Tolerante a fallas\estatus>
```

Conclusión

En este tercer programa tuvimos como reto realizar un servicio o script que nos permitiese informar en todo momento el estado de nuestra ejecución del programa principal, para en caso de algún cierre inesperado, este mismo script fuera capaz de levantarlo de nuevo, permitiendo así ensanchar nuestros programas para ser más permisible y tolerante a las fallas que se pueden presentar.

Para la realización del programa utilicé el módulo psutil propuesto por el docente, que permite detectar la ejecución de alguna tarea o proceso y a partir de allí informar el estado y levantar dicha app en caso de que esté cerrada, para lo que opté por utilizar os.system e ingresar en el comando la instrucción que ejecute el programa principal.

Sin duda este programa enriquece mis conocimientos ya que realicé un análisis previo, lo que me permite aclarar nuevas dudas en ámbitos de la tolerancia a fallas y mejorar mis habilidades de programación; Espero me ayude a seguir mejorando.

Bibliografías

(S/f). Tecnobillo.com. Recuperado el 28 de febrero de 2022, de <https://tecnobillo.com/sections/python-en-windows/servicios-windows-python/servicios-windows-python.html>

Psutil documentation — psutil 5.9.1 documentation. (s/f). Readthedocs.io. Recuperado el 28 de febrero de 2022, de <https://psutil.readthedocs.io/en/latest/>

LINK AL REPOSITORIO:

<https://github.com/UlisesVallejo/Serializaci-n-Application-checkpointing-.git>