# ESTATUS

COMPUTACIÓN TOLERANTE A FALLAS

DR. MICHEL EMMANUEL LOPEZ FRANCO

SECCIÓN D06

L-MI. 1:00-3:00PM

**UNIVERSIDAD DE GUADALAJARA,**
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERIAS

MURILLO CORTES, JEANETTE

# ESTATUS. RESTAURAR EL ESTADO

## PREINSTALACIONES

Para el desarrollo de la práctica, es recomendable:

- Tener 'psutil' instalado. Se puede instalar mediante el comando:
  `python -m pip install psutil`
- Tener descargado NSSM, link de descarga: https://nssm.cc/download además de seguir el tutorial de instalación del link https://tecnobillo.com/sections/python-en-windows/servicios-windows-python/servicios-windows-python.html

## DESARROLLO DE LA PRÁCTICA

Los servicios permiten realizar acciones que son ajenas al usuario, de manera que pueden ser utilizados para comprobar el estado de una aplicación o ejecutar programas en segundo plano sin que dependan de la interacción del usuario.
Tomando lo anterior en cuenta, en el desarrollo de la práctica se plantea que una aplicación permanezca abierta o ejecutandose mediante un servicio para que aún cuando haya ocurrido un error y tenga que cerrarse, se reabra la aplicación automáticamente.
Para ello, primeramente se creó el script para la aplicación que se iba a utilizar, en el cuál se desarrolla código que comprueba constantemente si el estado de la aplicación esta activo o no.
La aplicación en si hace uso de la técnica de serialización y desearialización con Pickle, donde si el programa es cerrado de manera inesperada, se puede continuar utilizandose en donde se quedó luego de abrirlo. De esta manera, si ocurre un error en la aplicación y se cierra, pero mediante el servicio se vuelve a abrir, la aplicación puede continuar en uso.

```
# Verificacion del estado de la
aplicación
```

*Importar librerias psutil y sys*

```
import sys
import psutil
```

*Uso de psutil*

```
def revisar_argumentos():
# Evitar errores
   if len(sys.argv)== 1:
     print('Este programa no
funciona sin argumentos')
     sys.exit(0)

def objetivos():
# Mediante argumentos se recibe
el nombre del proceso que se va
a interceptar
   targets = sys.argv[1:]
   i = 0
   while i<len(targets):
     if not
targets[i].endswith('.exe'):
# Para aceptar argumentos con .exe
        targets[i]=targets[i]+'
.exe'
     i += 1
   return targets
```

*Uso de psutil*

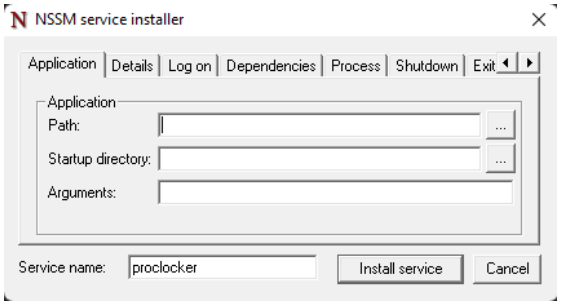
```
def bloquear(target):
   for proc in
psutil.process_iter():
      if proc.name().lower()
==target.lower():
         proc.kill() # Mata el
proceso
if __name__ == '__main__':
    revisar_argumentos()
    targets = objetivos()
```

```
while True:
    for target in targets:
        bloquear(target)
```

Como se puede observar, en el el código del script que se encuentra a la derecha, primeramente se monitorea si hay errores en la aplicación mediante el paso de argumentos, además de interceptar mediante estos argumentos el nombre de la aplicación a la que se va a aplicar el servicio, utilizando la función 'endswith()' para que automaticamente le añada el '.exe' y se pueda reabrir el script. También se añade una funcion que es la encargada de matar el proceso, la cuál es llamada al ultimo de los procesos.
Como la aplicación se puede monitorear mediante la creación de un Servicio, se hizo uso de The Non-Sucking Service Manager (NSSM), el cuál permite supervisar el servicio en ejecución y reiniciarlo si muere.
Para esto, después de la descarga de NSSM y de agregarlo a la carpeta custom-services (como se comenta en el link), se debe de compilar el archivo nssm en la linea de comandos con los comandos:

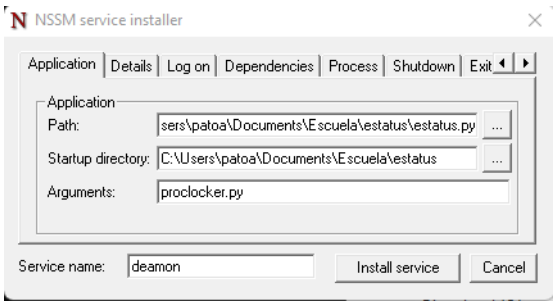
```
cd C:\custom-services
nssm.exe install proclocker
```

Esta acción no va a abrir un formulario, el cuál solicita la siguiente información del servicio (como se muestra en la imagen de la derecha):




- 'Path': la ruta del intérprete de Python
- 'Startup directory': directorio donde esta *C:\custom-services*
- 'Arguments': estos no son necesarios para la práctica, pero pueden utilizarse de otras maneras
- Nombre del servicio

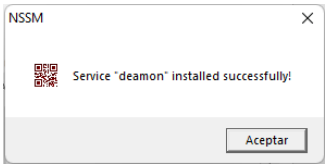
Una vez escrita dicha información, al terminar, se da click en el botón de 'Instalar el servicio'. Para la práctica, se desarrolló el servicio 'deamon', como se observa, y se añadió la información necesaria de la acitivad. De esta manera, en la linea de comandos debe aparecer un mensaje que comenta que se ha realizado el servicio de manera ex itosa.



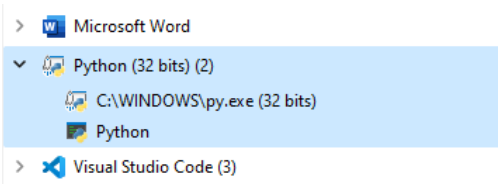

Por último, para iniciar dicho servicio, se debe escribir en el comand line:

```
nssm.exe start deamon
```

Esto debe de iniciar el servicio y devolver un mensaje que declara que se completó correctamente.



Para observar que la aplicación se ejecuta aun después de matarla o de un error, podemos observarla en el administrador de tareas, como se muestra en la imagen de la derecha.




## CONCLUSIÓN

Administrar el estado de una aplicación nos permite implementar un sistema para supervisar la condición en la que buscamos que se mantenga una aplicación. Para el desarrollo de la práctica, se buscaba que aunque hubiera un fallo en el sistema o un error que creara una falla y matara el programa, la aplicación pudiera automaticamente resolver este problema y mantener viva la aplicación. Esto nos permite observar .

NSSM

PYTHON

## BLIBLIOGRAFÍA


- *"Crear Servicios para Windows con Python"* (12 de Marzo, 2022) (tecnobillo.com) https://tecnobillo.com/sections/python-en-windows/servicios-windows-python/servicios-windows-python.html
- *"NSSM - the Non-Sucking Service Manager"* https://nssm.cc/download
- Suárez Lamadrid, Alejandro y Suárez Jiménez, Antonio. *"Python 3 para impacientes: Threading: programación con hilos (I)"* (python-para-impacientes.blogspot.com) https://python-para-impacientes.blogspot.com/2016/12/threading-programacion-con-hilos-i.html