COMPUTACIÓN TOLERANTE A FALLAS

Ingeniería en computación

López Franco Michel Emanuel

CUCEI

CORREO: juan.guillen9059@alumnos.udg.mx

ALUMNO: Guillén García Juan Andrés

TELEFONO: +52 33 3821 0156

CÓDIGO: 220790598

SECCIÓN: D06

CICLO: 24A

Fecha: 18 / Febrero / 2024

Índice

Introducción

¿Qué es un hilo?

Es una unidad básica de utilización de CPU, la cual contiene un id de hilo, su propio program counter, un conjunto de registros, y una pila; que se representa a nivel del sistema operativo con una estructura llamada TCB (thread control block). Los hilos comparten con otros hilos que pertenecen al mismo proceso la sección de código, la sección de datos, entre otras cosas. Si un proceso tiene múltiples hilos, puede realizar más de una tarea a la vez (esto es real cuando se posee más de un CPU).

Ventajas:

- Respuesta: el tiempo de respuesta mejora, ya que el programa puede continuar ejecutándose, aunque parte de él esté bloqueado.
- Compartir recursos: los hilos comparten la memoria y los recursos del proceso al que pertenecen, por lo que se puede tener varios hilos de ejecución dentro del mismo espacio de direcciones.
- Utilización múltiples CPUs: permite que hilos de un mismo proceso ejecuten en diferentes CPUs a la vez. En un proceso mono-hilo, un proceso ejecuta en una única CPU, independientemente de cuantas tenga disponibles.

¿Para que sirve un Demonio en programación?

Son procesos que se ejecutan continuamente en segundo plano y realizan funciones necesarias para otros procesos. TCP/IP (Transmission Control Protocol/Internet Protocol) proporciona daemons para implementar determinadas funciones en el sistema operativo. Estos daemons son procesos en segundo plano que se ejecutan sin interrumpir otros procesos (a menos que esto forme parte de la función del daemon).

¿Qué permite la programación concurrente?

La programación concurrente permite desarrollar software que ejecuta eventos o circunstancias que están sucediendo o existen al mismo tiempo. Los módulos concurrentes interactúan enviándose mensajes entre sí.

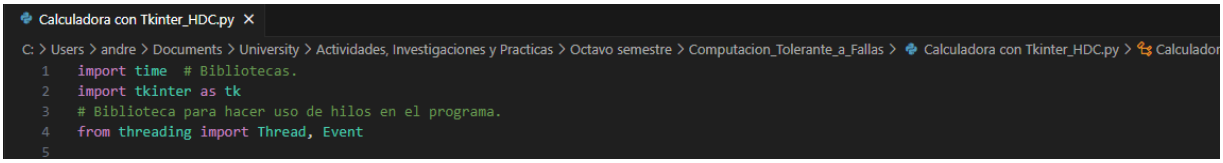
Objetivo

Realizar un programa haciendo uso de hilos, procesos, demonios y concurrencia. También el revisar los enlaces adjuntos en la actividad de Classroom para entender mejor el uso de los conceptos previamente comentados.

Desarrollo

Calculadora con hilos, demonios y concurrencia.

Como se puede observar en la siguiente imagen están las importaciones de las bibliotecas que voy a utilizar, donde con la que me tuve que documentar fue con la de threading que es básicamente para el uso de hilos en Python y la importante para esta actividad.



A continuación esta la clase “Calculadora_B”, en donde de manera resumida defino los ajustes de la interfaz gráfica de la calculadora con Tkinter, además de definir el funcionamiento de los botones de entrada y su acomodo.

```
9  class Calculadora_B(tk.Tk):
10     def __init__(self):  # Método inicializador de la clase
11         super().__init__()
12         self.title("Calculadora")  # Titulo del programa.
13         self.geometry("225x290")  # Ajuste del tamaño de la ventana del programa.
14         self.configure(bg="black") # Fondo negro.
15
16         # Configuración de la entrada de texto y botones.
17         self.entry = tk.Entry(self, width=34, bg="white", fg="black", bd=5)  # Cuadro de texto con fondo blanco y letras negras.
18         self.entry.grid(row=0, column=0, columnspan=4, padx=5, pady=5)
19         button_bg = "gray"  # Color de boton.
20         button_fg = "white"  # Color del texto de los botones.
21
22         buttons = [
23             ("7", 1, 0), ("8", 1, 1), ("9", 1, 2), ("/", 1, 3), # Acomodo de los botones
24             ("4", 2, 0), ("5", 2, 1), ("6", 2, 2), ("*", 2, 3), # de la calculadora.
25             ("1", 3, 0), ("2", 3, 1), ("3", 3, 2), ("-", 3, 3),
26             ("0", 4, 0), (".", 4, 1), ("=", 4, 2), ("+", 4, 3),
27             ("C", 5, 0, 4)
28         ]
29
30         for button in buttons:
31             if button[0] == "C": # Este apartado es unicamente para el boton "Clean".
32                 text, row, col, colspan = button
33                 btn = tk.Button(self, text=text, width=20, height=2, bg=button_bg, fg=button_fg, command=lambda t=text: self.Detector_B(t))
34                 btn.grid(row=row, column=col, columnspan=colspan, padx=5, pady=5)
35             else: # Todos los demas botones.
36                 text, row, col = button
37                 btn = tk.Button(self, text=text, width=5, height=2, bg=button_bg, fg=button_fg, command=lambda t=text: self.Detector_B(t))
38                 btn.grid(row=row, column=col, padx=5, pady=5)
39
40         self.running = False
41         self.result = None
42         # Hilo para realizar el cálculo en segundo plano y evento que lo detiene.
43         self.thread = None
44         self.stop_event = Event()
```

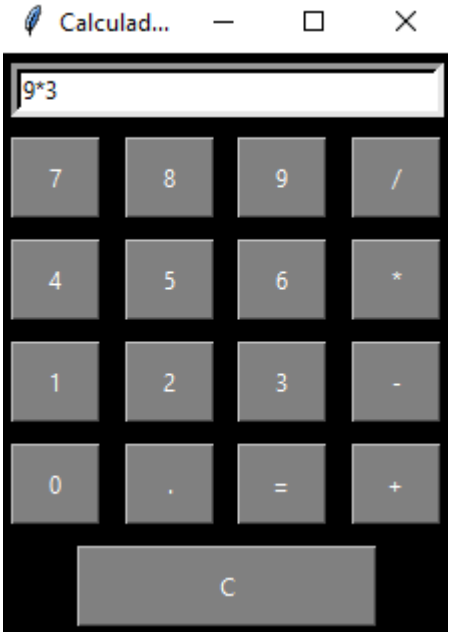
En la imagen anterior también se puede apreciar al final el uso de “self.thread = none”, dicho apartado es para que el programa realice en segundo plano el cálculo y después de esta instrucción se encuentra un evento el cual detiene el hilo. En la siguiente imagen se puede apreciar el método “Detector_B”, este funciona de manera en que cuando la persona hace Click en alguno de los botones donde por medio de un Switch en Python selecciona si el botón es “=, C, operador y valores). Como se puede apreciar en la imagen se encuentra la siguiente línea de código “self.thread = Thread(target=self.Resultado, args=(current_text,))”, en esta parte se crea un hilo que ejecutara el resultado junto con su objetivo y asi con los argumentos pasar el texto al hilo para asi ser procesado. Luego se declara el hilo como demonio con “self.thread.daemon = True” para asi hacer que el hilo se detenga de manera automática al cerrar el programa y por último se inicia el hilo con la configuración que se le asigno previamente.

```
46      def Detector_B(self, text):
47          current_text = self.entry.get()  # Captura los valores.
48          if text == '=':
49              if not self.running:  # Si no se está ejecutando ningún cálculo actualmente.
50                  self.thread = Thread(target=self.Resultado, args=(current_text,))  # Hace uso de un hilo para obtener el resultado.
51                  self.thread.daemon = True  # Convertimos el hilo a demonio.
52                  self.thread.start()  # Inicia el hilo.
53          elif text == 'C':
54              self.entry.delete(0, tk.END)  # Borra todo en la pantalla.
55          else:
56              self.entry.insert(tk.END, text)  # Se agrega el texto que indique el boton.
```

En la siguiente imagen se encuentra el método del resultado que funciona en conjunto con el método “Detector_B”, donde por medio de un hilo separado se ejecuta la operación mientras que el usuario puede seguir interactuando con la calculadora. A su vez le agregue un “time.sleep” de 5 segundos para que de tiempo a modificar los datos a la vez que la calculadora está realizando la operación. Una vez terminada la explicación de esta parte se encuentra el final del programa en donde muestra el resultado en la pantalla por medio de una actualización de la misma y continuando detiene el hilo y se cierra la ventana.

```
58      def Resultado(self, text):
59          self.running = True
60          time.sleep(5) # Aparenta que se tarda en ejecutar la operacion.
61          try: # Validacion para que el programa no tenga fallas.
62              result = eval(text)  # Evalúa la expresión matemática ingresada
63              self.result = result  # Guarda el resultado.
64          except Exception as e:
65              self.result = "No se pudo realizar la operacion" # Muestra mensaje de error.
66          self.running = False
67          self.A_Pantalla()
68
69      def A_Pantalla(self):  # Muestra el resultado en el cuadro de texto.
70          if self.result is not None:  # Cuando el resultado es posible.
71              self.entry.delete(0, tk.END)  # Borra todo el texto.
72              self.entry.insert(tk.END, str(self.result))  # Inserta el resultado.
73
74      def Cerrar_P(self):  # Método que se ejecuta cuando se cierra la ventana.
75          self.stop_event.set()  # Evento que detiene el hilo.
76          self.destroy()  # Destruye la ventana.
77
78  if __name__ == "__main__":
79      app = Calculadora_B()  # Inicia la ejecucucion del programa.
80      app.protocol("WM_DELETE_WINDOW", app.Cerrar_P)
81      app.mainloop() # Inicia un bucle en la interfaz.
```

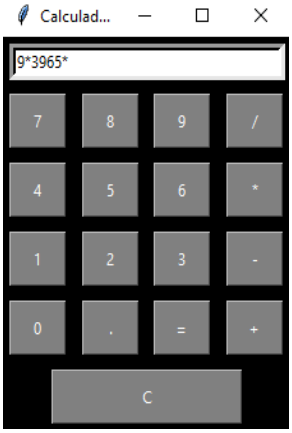
Captura del funcionamiento.

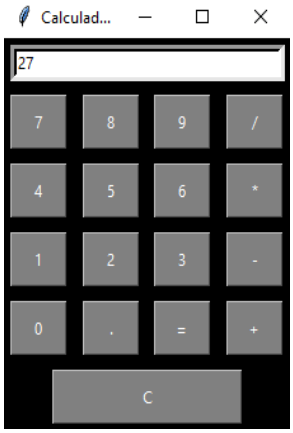


Representar el funcionamiento del programa por medio de capturas es bastante complicado por el hecho de que no se puede ver en tiempo real el propósito del programa. Asi que con las imágenes intentare hacer que se entienda el cómo funciona.

A la izquierda se puede observar la calculadora con los botones que contienen valor numérico, los operadores, el botón de resultado y el botón para limpiar la pantalla.

Entonces una vez yo le dé al botón de resultado, el programa enviará el cálculo a un hilo independiente, con un tiempo de espera de 5 segundos en donde el usuario podrá seguir interactuando con la interfaz y el cálculo seguirá siendo procesado para mostrar el resultado, sin afectar el resultado.





Con las imágenes de la izquierda trato de demostrar el hecho de que una vez que presiono el botón “=” el resultado no se ve alterado aunque el usuario siga presionando botones que agreguen números o operadores a la pantalla, esto debido a que los demonios en Python permiten seguir interactuando con la interfaz sin que se detenga el procesamiento en los hilos y como la operación base ya se había enviado al hilo independiente, la persona puede seguir presionando botones sin que existan cambios en los resultados de esa misma operación. Por lo cual en este programa se hace uso de hilos, demonios y la concurrencia al tratarse de la interacción de los métodos entre si dentro de la aplicación.

Carrera de caballos con procesos (Método First Come First Served).

Esta actividad la realicé en otra clase y decidí usarla para esta actividad por el hecho de que es un algoritmo de planificación de procesos, los cuales simulan una carrera de caballos en donde cada uno funciona de manera independiente y muestra los tiempos con los que cada caballo termino la carrera. El programa cuenta con un botón para iniciar el proceso y uno para reiniciar la carrera, el cual también está realizado en Python pero de una manera mucho mas extensa que la calculadora básica que coloque anteriormente.

main.py

Dentro de este archivo se manda llamar tanto la aplicación como la ventana de la interfaz.

```
from PySide6.QtWidgets import QApplication
from mainwindow import MainWindow
import sys



app = QApplication()
window = MainWindow()window.show()


sys.exit(app.exec_())
```

mainwindows.py

Dentro de este archivo se encuentran las declaraciones, funciones, llamadas e implementación del código para que este funciones correctamente por medio del algoritmo de planificación.

```
from PySide6.QtWidgets import QMainWindow
from PySide6.QtCore import Slot, QTimer
from ui_mainwindow import Ui_MainWindow
from PySide6.QtWidgets import QMainWindow
from random import randint
import time

from proceso import Proceso

class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow,self).__init__()
```

```
        self.t_total = 0      #contador de tiempo
        self.caballos = []
        self.espera = []
        self.terminados = []
        #declara caballos
        self.c1 = Proceso()
        self.c2 = Proceso()
        self.c3 = Proceso()
        self.c4 = Proceso()
        #nombrar caballos
        self.c1.nombre = 'Caballo 1'
        self.c2.nombre = 'Caballo 2'
self.c3.nombre = 'Caballo 3'
        self.c4.nombre = 'Caballo 4'
        #crear la ui
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        #Crear timer para mover la ui
        self.timer = QTimer(self)

        # concecion de botones con una funcion
        self.ui.restart_pushButton.clicked.connect(self.restart)
        self.ui.start_pushButton.clicked.connect(self.start)

    #Funcion para reiniciar el programa
    @Slot( )
    def restart(self):
        self.timer.stop()
        self.ui.info_label.clear()
        self.ui.posiciones_label.clear()
        #restablecer datos
        self.t_total = 0
        self.caballos = []
        self.espera = []
        self.terminados = []
        #declara caballos
        self.c1 = Proceso()
        self.c2 = Proceso()
        self.c3 = Proceso()
        self.c4 = Proceso()
        #nombrar caballos
        self.c1.nombre = 'Caballo 1'
        self.c2.nombre = 'Caballo 2'
        self.c3.nombre = 'Caballo 3'
        self.c4.nombre = 'Caballo 4'
        self.start()

    #Funcion para iniciar el programa
    @Slot( )
```

```
    def start(self):
#reiniciar posiciones de los caballos
        self.ui.caballo1_label.move(60,10)
        self.ui.caballo2_label.move(60,150)
        self.ui.caballo3_label.move(60,290)
        self.ui.caballo4_label.move(60,430)
        #asignar tiempo a acaballos
        self.c1.tiempo = randint(5,15)
        self.c2.tiempo = randint(5,15)
        self.c3.tiempo = randint(5,15)
        self.c4.tiempo = randint(5,15)

        self.c1.tiempo_fal = self.c1.tiempo
        self.c2.tiempo_fal = self.c2.tiempo
        self.c3.tiempo_fal = self.c3.tiempo
        self.c4.tiempo_fal = self.c4.tiempo

        #agragar al primer caballo al procesador
        self.c1.tiempo_lleg = self.t_total
        self.caballos.append(self.c1)

        #agregar los demas caballos al espera
        self.espera.insert(0,self.c2)
        self.espera.insert(0,self.c3)
        self.espera.insert(0,self.c4)

        #iniciamos el timer
        self.timer.timeout.connect(self.mover_caballo)
        self.timer.start(1000)

    @Slot( )
    def mover_caballo(self):
        self.t_total += 1

        if self.caballos[0].tiempo_fal > 0:
            print(self.caballos[0].tiempo_fal)
```

```
            print()
            self.caballos[0].tiempo_fal -= 1


            paso = 900 // self.caballos[0].tiempo
            if self.caballos[0].nombre == 'Caballo 1':
                x = self.ui.caballo1_label.x()
                y = self.ui.caballo1_label.y()
                self.ui.caballo1_label.move(x+paso,y)
            elif self.caballos[0].nombre == 'Caballo 2':
                x = self.ui.caballo2_label.x()
                y = self.ui.caballo2_label.y()
                self.ui.caballo2_label.move(x+paso,y)
            elif self.caballos[0].nombre == 'Caballo 3':
                x = self.ui.caballo3_label.x()
                y = self.ui.caballo3_label.y()
                self.ui.caballo3_label.move(x+paso,y)
            elif self.caballos[0].nombre == 'Caballo 4':
                x = self.ui.caballo4_label.x()
                y = self.ui.caballo4_label.y()
                self.ui.caballo4_label.move(x+paso,y)

            for pony in self.espera:
                pony.espera += 1

        elif len(self.espera) > 0:
            term = self.caballos.pop()
            term.fin = self.t_total
            self.terminados.append(term)

            new = self.espera.pop()
            new.tiempo_lleg = self.t_total - 1
            new.inicio = self.t_total
            self.caballos.append(new)

        else:
            term = self.caballos.pop()
            term.fin = self.t_total
            self.terminados.append(term)

            cad = ''
            cad2 = ''
            num = []
            for pony in self.terminados:
                cad += str(pony.nombre)
+'\t||'+str(pony.tiempo_lleg)+'\t\t||'+str(pony.tiempo)+'\t\t||'+str(pon
y.inicio)+'\t||'+str(pony.fin)+'\t||'+str(pony.espera)+'\n'
self.ui.info_label.setText('Proceso\t\t||TLlegada\t||Tiempo\t||Inicio\t|
|Fin\t||Espera\n'+cad)
            for pony in self.terminados:
```

```
                tup =(pony.tiempo,pony.nombre)
                num.append(tup)
            num.sort(key= lambda x: x[0])
            for x in num:
                cad2 += x[1] + ' - ' + str(x[0]) + '\n'
            self.ui.posiciones_label.setText(cad2)
            self.timer.stop()
```

proceso.py

Dentro de este archivo se encuentran las declaración de todas las variables del programa inicializadas en cero.

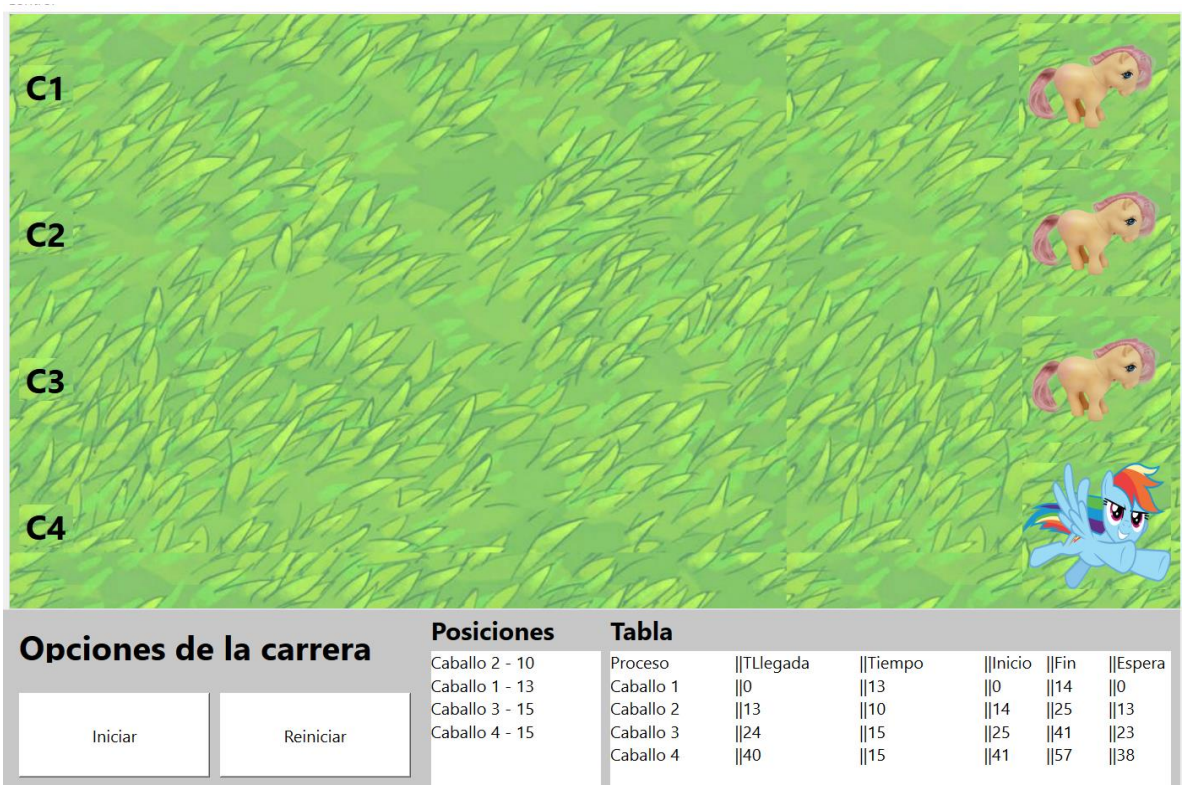
```
class Proceso:
    def __init__(self):
        self.nombre = ''
        self.tiempo_lleg = 0
        self.tiempo = 0
        self.tiempo_fal = 0
        self.inicio = 0
        self.fin = 0
        self.espera = 0
```

Captura del funcionamiento.

En esta captura se muestra la interfaz gráfica con tres caballos y raimbow dash, asi tratando de simular un hipódromo. Aquí todos los valores están en cero y aun no se presiona el botón iniciar.

Las siguientes capturas muestran el proceso del programa al presionar el botón iniciar, por obvias razones no se notará el movimiento tal cual pero para darse una idea se pondrán varias imágenes.

En la siguiente captura se puede ver el resultado final y los datos de los caballos junto con las estadísticas que estos obtuvieron a lo largo del recorrido.



Aquí se puede apreciar que al reiniciar la carrera los datos y resultados de esta cambiaron, asi siendo totalmente aleatorio los tiempos de los procesos cada vez que se iniciaba una nueva carrera.

Conclusión

Para completar esta práctica decidí realizar una calculadora bastante sencilla la cual usa un hilo para guardar la operación y demonios que permiten el seguir interactuando con el programa mientras se calcula el resultado de la operación realizada, en un inicio quería plantear la actividad en un solo código pero se me complico el uso de multiprocesos e hilos en el mismo programa asi que para realizar la parte de los procesos que se solicitaba en la actividad decidí el usar un programa que había realizado en otra materia de las que estoy cursando en este semestre, a pesar de que los caballos están graciosos el programa funciona perfectamente con procesos independientes y cada uno de los caballos va avanzando en la ventana del programa hasta llegar al final usando el método FCFS.

En si el realizar este programa no fue tan complicado por el hecho de que el programa que realice especialmente para esta actividad era un tanto sencillo pero cumplía con parte de los requisitos, con respecto a la idealización de hacer un programa con todo lo que se pedía me fue un tanto complicado por el hecho de estar haciendo la actividad de los caballitos que use para dar el ejemplo de la parte de los procesos, porque esa actividad si fue bastante confusa y complicada de realizar, espero que la tarea valga lo mismo a pesar de que use dos códigos para entregarla en vez de solo plantear uno como la descripción de la actividad decía.

Bibliografía

Hilos. (n.d.). https://www.fing.edu.uy/tecnoinf/maldonado/cursos/so/material/teo/so05-hilos.pdf

Daemons TCP/IP. (n.d.). Www.ibm.com. Retrieved February 19, 2024, from https://www.ibm.com/docs/es/aix/7.1?topic=protocol-tcpip-daemons

threading — Paralelismo basado en hilos — documentación de Python - 3.8.18. (n.d.). Docs.python.org. Retrieved February 19, 2024, from https://docs.python.org/es/3.8/library/threading.html

Concurrency in Python: Threading, Processes, and Asyncio - StatusNeo. (2023, May 3). https://statusneo.com/concurrency-in-python-threading-processes-and-asyncio/

Como hacer un demonio en python. (n.d.). Javisantana.com. Retrieved February 19, 2024, from https://javisantana.com/2010/03/21/como-hacer-un-demonio-en-python.html

An Introduction to Scaling Distributed Python Applications. (n.d.). Educative. Retrieved February 19, 2024, from https://www.educative.io/blog/scaling-in-python