Universidad de Guadalajara.

Centro Universitario de Ciencias Exactas e Ingenierías.

DIVISIÓN DE TECNOLOGÍAS PARA LA INTEGRACIÓN CIBER-HUMANA.

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES.

TEMA: Practica1

NOMBRE DE LOS ESTUDIANTES:

Padilla Perez Jorge Daray

Juan Jesús Sámano Juárez

Ernesto Macias Flores

NOMBRE DE LA MATERIA: Seminario Sistemas Operativos

SECCION: D04

CICLO ESCOLAR: 2024-A

NOMBRE DEL PROFESOR: Julio Esteban Valdés López

Table off Contents

Introducción:

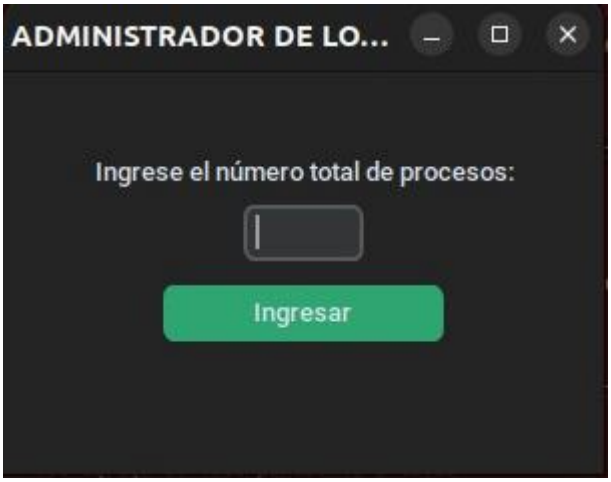
En esta practica inicial, se presenta el problema del procesamiento por lotes, en el cual se simulan estos empezando por pedirle al usuario que ingrese los procesos que quiere realizar en este caso los lotes están topados por 4 procesos cada uno, los cuales llevan la información del programador, la operación que realiza, el tiempo estimado, y el número del programa.

Seguido de eso, se muestra en pantalla el lote que se está ejecutando con sus datos, otra pantalla que muestra en que proceso va, cuánto tiempo lleva y cuantos faltan. Otra pantalla con los procesos terminados y para finalizar un contador global en el cual se incluye el tiempo total que se lleva trabajando en la aplicación.

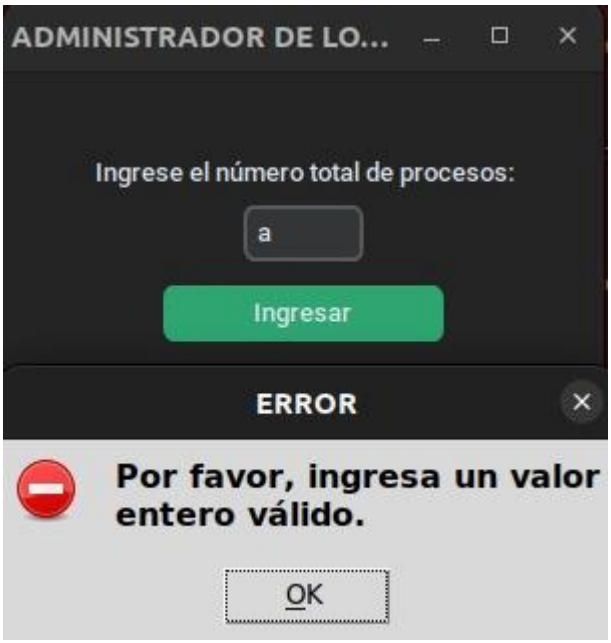
También se presentan capturas de las ejecuciones del programa funcionando de manera correcta de los momentos que creemos cruciales, el código fuente que se utilizó para el desarrollo del programa, y por ultimo las conclusiones personales de cada integrante del equipo.

Capturas de pantalla

Administrador de los lotes: Parte inicial del programa en donde se pone el numero de procesos que tendrá.



En caso de no introducir un numero entero esta validado para no proceder y manda un mensaje de error.



Como este:

Llenado de los lotes:

Aquí se puede apreciar la manera correcta de llenar el formulario:

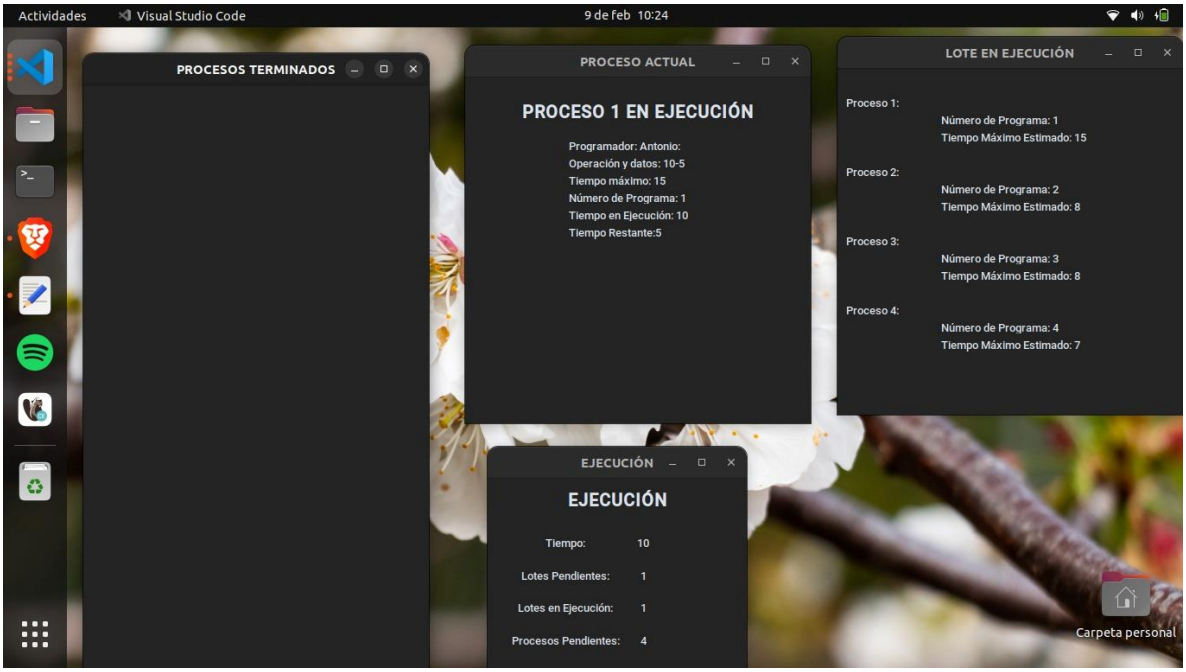
Aquí se aprecia la interfaz que esta tiene:

Cualquier cosa que no se llene correctamente manda mensaje de error y no permite continuar con el proceso

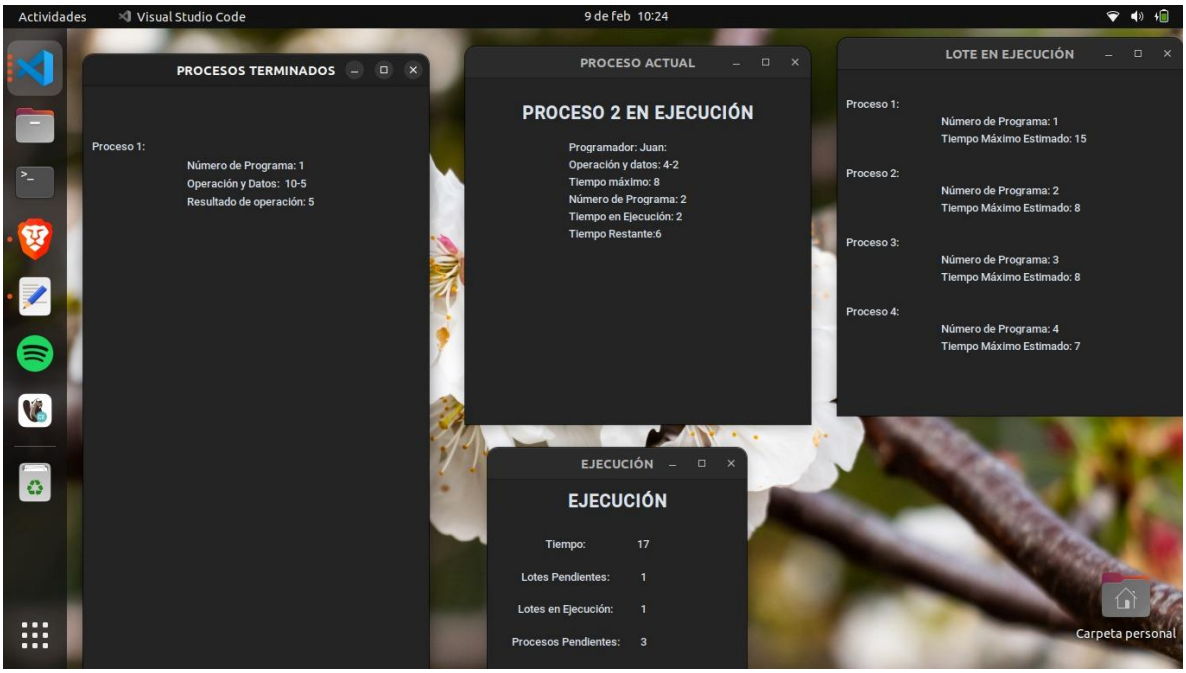
Tanto como la operación como el tiempo de espera máximo:

Corrida con 5 procesos:

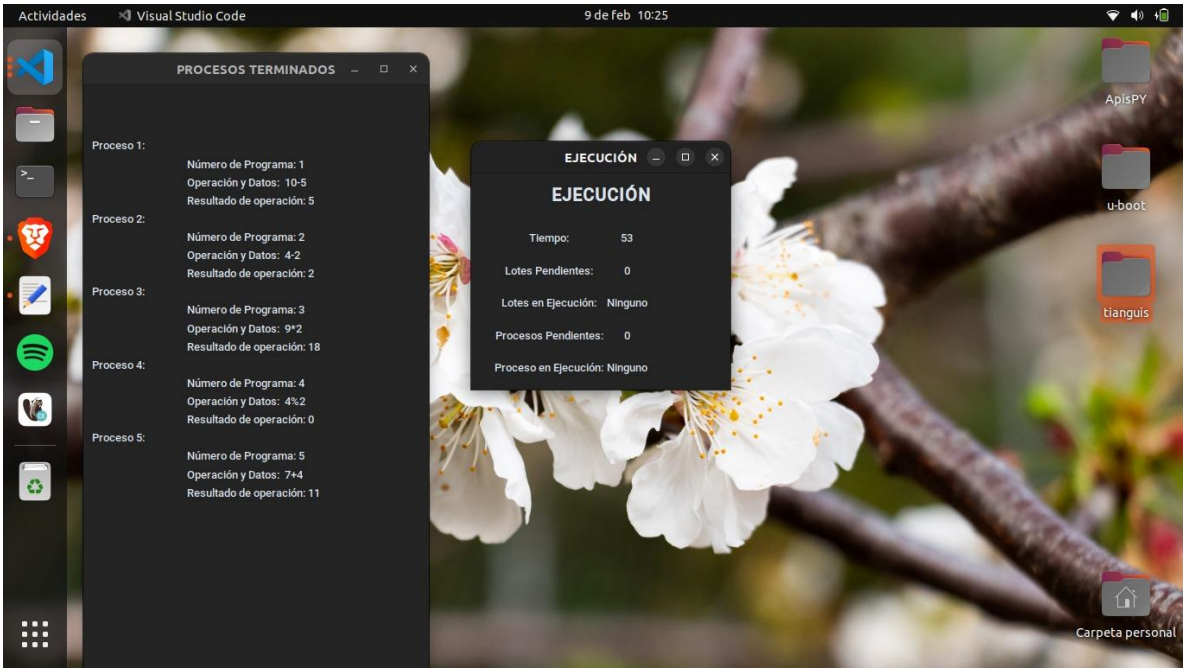
Primer proceso en ejecución en el que se muestran 4 ventanas en las cuales aparece el proceso actual, el cual se está procesando junto con sus datos, en otra ventana el lote que está en ejecución y los procesos que está llevando a cabo, otra ventana con la ejecución de los procesos que tiene el tiempo de estos, cuantos lotes y procesos faltan etc. Del lado izquierdo por último tenemos los procesos terminados en este caso no hay nada porque todavía no termina ninguno.



Aquí ya termino el primer proceso, este procede a agregarse a la ventana procesos terminados.



Por ultimo solo queda la ventana de ejecución que contiene el tiempo global actualizada sin lotes ni nada pendiente, y los procesos terminados para poder ver los resultados.



Código Fuente

Vistas

Captura.py

```
import customtkinter as ctk
import tkinter as tk
from utils import validateEntero

#Captura de Procesos Programa
class Captura(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("ADMINISTRADOR DE LOTES")
        self.geometry("500x350")
        self.captura = {}

        self.titulo = ctk.CTkLabel(self, text="CAPTURA DE PROCESOS",
font=ctk.CTkFont(size=20, weight="bold"))
        self.titulo.place(relx=0.5, rely=0.1, anchor=ctk.CENTER)
```

```
        self.labelProgramador = ctk.CTkLabel(self, text="Nombre del
programador:", fg_color="transparent")
        self.labelProgramador.place(relx=0.3, rely=0.3, anchor=ctk.CENTER)
        self.inputProgramador = ctk.CTkEntry(self,
placeholder_text="Nombre...",width=200)
        self.inputProgramador.place(relx=0.7, rely=0.3, anchor=ctk.CENTER)

        self.labelOperacion = ctk.CTkLabel(self, text="Operación a
realizar:", fg_color="transparent")
        self.labelOperacion.place(relx=0.3, rely=0.45, anchor=ctk.CENTER)
        self.inputOperacion = ctk.CTkEntry(self,
placeholder_text="Operación...",width=200)
        self.inputOperacion.place(relx=0.7, rely=0.45, anchor=ctk.CENTER)

        self.labelTiempo = ctk.CTkLabel(self, text="Tiempo máximo de
espera:", fg_color="transparent")
        self.labelTiempo.place(relx=0.3, rely=0.6, anchor=ctk.CENTER)
        self.inputTiempo = ctk.CTkEntry(self,
placeholder_text="Tiempo...",width=200)
        self.inputTiempo.place(relx=0.7, rely=0.6, anchor=ctk.CENTER)

        self.btnCapturar = ctk.CTkButton(self, text="Capturar",
command=lambda: self.validateCampos())
        self.btnCapturar.place(relx=0.5, rely=0.90, anchor=ctk.CENTER)

    def validateCampos(self):
        operadores = ['+', '-', '*', '/', '%', '^']
        if not any(op in self.inputOperacion.get() for op in operadores):
            tk.messagebox.showerror("ERROR", "Operación inválida.")
            self.inputOperacion.delete(0,"end")
            return
        if not validateEntero(self.inputTiempo.get()):
            self.inputTiempo.delete(0,"end")
            return
        if self.inputTiempo.get() == '0':
            tk.messagebox.showerror("ERROR", "El tiempo debe ser mayor a
0.")
            self.inputTiempo.delete(0,"end")
            return
        self.getCapturaData()

    def getCapturaData(self):
        captura = {
            'programador' : self.inputProgramador.get(),
            'operacion' : self.inputOperacion.get().replace(" ","),
```

```
            'tiempo_maximo' : int(self.inputTiempo.get())
        }
        self.captura = captura
        self.destroy()


        self.labelProgramador = ctk.CTkLabel(self, text="Nombre del
programador:", fg_color="transparent")
        self.labelProgramador.place(relx=0.3, rely=0.3, anchor=ctk.CENTER)
        self.inputProgramador = ctk.CTkEntry(self,
placeholder_text="Nombre...",width=200)
        self.inputProgramador.place(relx=0.7, rely=0.3, anchor=ctk.CENTER)

        self.labelOperacion = ctk.CTkLabel(self, text="Operación a
realizar:", fg_color="transparent")
        self.labelOperacion.place(relx=0.3, rely=0.45, anchor=ctk.CENTER)
        self.inputOperacion = ctk.CTkEntry(self,
placeholder_text="Operación...",width=200)
        self.inputOperacion.place(relx=0.7, rely=0.45, anchor=ctk.CENTER)

        self.labelTiempo = ctk.CTkLabel(self, text="Tiempo máximo de
espera:", fg_color="transparent")
        self.labelTiempo.place(relx=0.3, rely=0.6, anchor=ctk.CENTER)
        self.inputTiempo = ctk.CTkEntry(self,
placeholder_text="Tiempo...",width=200)
        self.inputTiempo.place(relx=0.7, rely=0.6, anchor=ctk.CENTER)

        self.btnCapturar = ctk.CTkButton(self, text="Capturar",
command=lambda: self.validateCampos())
        self.btnCapturar.place(relx=0.5, rely=0.90, anchor=ctk.CENTER)

    def validateCampos(self):
        operadores = ['+', '-', '*', '/', '%', '^']
        if not any(op in self.inputOperacion.get() for op in operadores):
            tk.messagebox.showerror("ERROR", "Operación inválida.")
            self.inputOperacion.delete(0,"end")
            return
        if not validateEntero(self.inputTiempo.get()):
            self.inputTiempo.delete(0,"end")
            return
        if self.inputTiempo.get() == '0':
            tk.messagebox.showerror("ERROR", "El tiempo debe ser mayor a
0.")
            self.inputTiempo.delete(0,"end")
            return
```

```
        self.getCapturaData()

    def getCapturaData(self):
        captura = {
            'programador' : self.inputProgramador.get(),
            'operacion' : self.inputOperacion.get().replace(" ",""),
            'tiempo_maximo' : int(self.inputTiempo.get())
        }
        self.captura = captura
        self.destroy()
```

inicio.py

```
import customtkinter as ctk
from utils import validateEntero


#Inicio Programa (Procesos)
class Inicio(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("ADMINISTRADOR DE LOTES")
        self.geometry("300x200")
        self.procesos = 0

        self.labelProcesos = ctk.CTkLabel(self, text="Ingrese el número
total de procesos:", fg_color="transparent")
        self.labelProcesos.place(relx=0.5, rely=0.25, anchor=ctk.CENTER)

        self.inputProcesos = ctk.CTkEntry(self,
placeholder_text="",width=60)
        self.inputProcesos.place(relx=0.5, rely=0.4, anchor=ctk.CENTER)

        self.btnProcesos = ctk.CTkButton(self, text="Ingresar",
command=lambda: self.setProcesos())
        self.btnProcesos.place(relx=0.5, rely=0.6, anchor=ctk.CENTER)

    def setProcesos(self):
        if validateEntero(self.inputProcesos.get()):
            self.procesos = int(self.inputProcesos.get())
            self.destroy()
        else:
            self.inputProcesos.delete(0,"end")
```

simulacion.py

```
import customtkinter as ctk
import tkinter as tk
from Vistas.simulacionLotes import SimulacionLote
from Vistas.simulacionProcesos import SimulacionProcesos
from utils import
getLotesPendientes,deleteProcesos,getProcesosPendientes,deleteLotes,getLotes
Ejecucion,getProcesoEjecucion, getProcesos, getProcesoEjecucionObjeto

#Simualación de Procesos
class Simulacion(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("EJECUCIÓN")
        self.geometry("300x250")
        self.tiempo = -1
        self.tiempo_proceso = -1

        global pantallaSimulacionLote,pantallaSimulacionProceso

        lotes_pendientes = getLotesPendientes()
        lotes_ejecucion = getLotesEjecucion()
        procesos_pendientes = getProcesosPendientes()
        procesos_ejecucion = getProcesoEjecucion()


        self.titulo = ctk.CTkLabel(self, text="EJECUCIÓN",
font=ctk.CTkFont(size=20, weight="bold"))
        self.titulo.place(relx=0.5, rely=0.1, anchor=ctk.CENTER)

        self.labelTiempo= ctk.CTkLabel(self, text="Tiempo:",
fg_color="transparent")
        self.labelTiempo.place(relx=0.3, rely=0.3, anchor=ctk.CENTER)
        self.labelTiempoV= ctk.CTkLabel(self, text="0" ,
fg_color="transparent")
        self.labelTiempoV.place(relx=0.6, rely=0.3, anchor=ctk.CENTER)

        self.labelLotesPendientes = ctk.CTkLabel(self, text="Lotes
Pendientes:", fg_color="transparent")
        self.labelLotesPendientes.place(relx=0.3, rely=0.45,
anchor=ctk.CENTER)
        self.labelLotesPendientesV= ctk.CTkLabel(self, text=lotes_pendientes
, fg_color="transparent")
```

```
        self.labelLotesPendientesV.place(relx=0.6, rely=0.45,
anchor=ctk.CENTER)

        self.labelLotesEjecucion = ctk.CTkLabel(self, text="Lotes en
Ejecución:", fg_color="transparent")
        self.labelLotesEjecucion.place(relx=0.3, rely=0.6,
anchor=ctk.CENTER)
        self.labelLotesEjecucionV= ctk.CTkLabel(self, text=lotes_ejecucion ,
fg_color="transparent")
        self.labelLotesEjecucionV.place(relx=0.6, rely=0.6,
anchor=ctk.CENTER)

        self.labelProcesosPendientes = ctk.CTkLabel(self, text="Procesos
Pendientes:", fg_color="transparent")
        self.labelProcesosPendientes.place(relx=0.3, rely=0.75,
anchor=ctk.CENTER)
        self.labelProcesosPendientesV= ctk.CTkLabel(self,
text=procesos_pendientes , fg_color="transparent")
        self.labelProcesosPendientesV.place(relx=0.6, rely=0.75,
anchor=ctk.CENTER)

        self.labelProcesosEjecucion = ctk.CTkLabel(self, text="Proceso en
Ejecución:", fg_color="transparent")
        self.labelProcesosEjecucion.place(relx=0.3, rely=0.9,
anchor=ctk.CENTER)
        self.labelProcesosEjecucionV= ctk.CTkLabel(self,
text=procesos_ejecucion , fg_color="transparent")
        self.labelProcesosEjecucionV.place(relx=0.6, rely=0.9,
anchor=ctk.CENTER)

        self.actualizar_tiempo()

        pantallaSimulacionLote = SimulacionLote(getProcesos())
        pantallaSimulacionLote.mainloop()
        pantallaSimulacionProceso =
SimulacionProcesos(getProcesoEjecucionObjeto())
        pantallaSimulacionProceso.mainloop()


    def actualizar_tiempo(self):
        self.tiempo += 1
        self.tiempo_proceso += 1
        self.labelTiempoV.configure(text=str(self.tiempo))

        self.actualizar_procesos()
```

```
        self.actualizar_lotes()

        self.after(1000, self.actualizar_tiempo)

    def actualizar_procesos(self):
        procesos_pendientes = getProcesosPendientes()
        proceso_ejecucion = getProcesoEjecucion()
        global pantallaSimulacionProceso

        self.labelProcesosPendientesV.configure(text=str(procesos_pendientes
))
        self.labelProcesosEjecucionV.configure(text=str(proceso_ejecucion))
        borro_proceso = deleteProcesos(self.tiempo_proceso)
        if borro_proceso:
            print("OBEJTOOOO")
            if getProcesoEjecucionObjeto():
                pantallaSimulacionProceso.actualizar_procesos(getProcesoEjec
ucionObjeto())
            self.tiempo_proceso = 0

    def actualizar_lotes(self):
        lotes_pendientes = getLotesPendientes()
        lote_ejecucion = getLotesEjecucion()
        global pantallaSimulacionLote

        self.labelLotesPendientesV.configure(text=str(lotes_pendientes))
        self.labelLotesEjecucionV.configure(text=str(lote_ejecucion))
        borro_lotes = deleteLotes()
        if borro_lotes and getProcesos():
            pantallaSimulacionLote.actualizar_procesos(getProcesos())
            self.tiempo_proceso = 0
```

simulacionLotes.py

```
import customtkinter as ctk
import tkinter as tk

#Simualación de Procesos
class SimulacionLote(ctk.CTk):
    def __init__(self,procesos):
        super().__init__()

        self.title("LOTE EN EJECUCIÓN")
        self.geometry("400x400")
        self.procesos = procesos
        self.y_axis = 0.2

        self.renderizar_procesos()

    def renderizar_procesos(self):
        # Eliminar etiquetas anteriores
        for widget in self.winfo_children():
            widget.destroy()

        # Renderizar etiquetas de proceso
        self.y_axis = 0.2
        for i, proceso in enumerate(self.procesos):
            label_proceso = ctk.CTkLabel(self, text=f"Proceso {i+1}:",
fg_color="transparent")
            label_proceso.place(relx=0.1, rely=self.y_axis,
anchor=ctk.CENTER)

            label_programa = ctk.CTkLabel(self, text=f"Número de Programa:
{proceso.numero_programa}", fg_color="transparent")
            label_programa.place(relx=0.3, rely=self.y_axis+0.05,
anchor=ctk.W)

            label_tiempo = ctk.CTkLabel(self, text=f"Tiempo Máximo Estimado:
{proceso.tiempo_maximo}", fg_color="transparent")
            label_tiempo.place(relx=0.3, rely=self.y_axis+0.1, anchor=ctk.W)

            self.y_axis += 0.2

    def actualizar_procesos(self, nuevos_procesos):
        self.procesos = nuevos_procesos
        self.renderizar_procesos()
```

SimulacionProcesos.py

```
import customtkinter as ctk
import tkinter as tk

#Simualación de Procesos
class SimulacionProcesos(ctk.CTk):
    def __init__(self,proceso):
        super().__init__()

        self.title("LOTE EN EJECUCIÓN")
        self.geometry("400x400")
        self.proceso = proceso
        self.y_axis = 0.2
        print("ENTRO A INSTANCIA")

        self.renderizar_procesos()


    def renderizar_procesos(self):
        # Eliminar etiquetas anteriores
        print("HOLAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA")

        for widget in self.winfo_children():
            widget.destroy()

        self.titulo = ctk.CTkLabel(self, text="PROCESO EN EJECUCIÓN",
font=ctk.CTkFont(size=20, weight="bold"))
        self.titulo.place(relx=0.5, rely=0.1, anchor=ctk.CENTER)

        # Renderizar etiquetas de proceso
        self.y_axis = 0.2
        label_programador = ctk.CTkLabel(self, text=f"Programador:
{self.proceso.programador}:", fg_color="transparent")
        label_programador.place(relx=0.1, rely=self.y_axis,
anchor=ctk.CENTER)

        label_operacion = ctk.CTkLabel(self, text=f"Operación y datos:
{self.proceso.operacion}", fg_color="transparent")
        label_operacion.place(relx=0.3, rely=self.y_axis+0.05, anchor=ctk.W)

        label_resultado_operacion = ctk.CTkLabel(self, text=f"Resultado de
operación: {self.evaluar_operacion(self.proceso.operacion)}",
fg_color="transparent")
        label_resultado_operacion.place(relx=0.3, rely=self.y_axis+0.1,
anchor=ctk.W)
```

```
        self.y_axis += 0.2

    def actualizar_procesos(self, nuevos_procesos):
        self.proceso = nuevos_procesos
        self.renderizar_procesos()

    def evaluar_operacion(self, operacion):
        try:
            resultado = eval(operacion)  # Intenta evaluar la operación
           # print(f"Proceso {self.num_programa}: Resultado de
{self.operacion}: {resultado}")
        except Exception as e:
            return False
        return resultado
```

simulacionTerminados.py

```
import customtkinter as ctk

# Simulación de Procesos
class SimulacionProcesosTerminados(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("PROCESOS TERMINADOS")
        self.geometry("400x800")
        self.procesos_terminados = []  # Lista para almacenar procesos
terminados
        self.y_axis = 0.1


    def renderizar_procesos(self):
        print("Aqui en renderizado")
        # Eliminar etiquetas anteriores
        for widget in self.winfo_children():
            if not isinstance(widget, ctk.CTkCanvas):
                widget.destroy()
        print("paso o c murio")
        self.y_axis = 0.1
        # Renderizar etiquetas de proceso para todos los procesos en la
lista
        for proceso in self.procesos_terminados:
            label_proceso = ctk.CTkLabel(self, text=f"Proceso
{proceso.no_proceso}:", fg_color="transparent")
            label_proceso.place(relx=0.1, rely=self.y_axis,
anchor=ctk.CENTER)
```

```
            label_programa = ctk.CTkLabel(self, text=f"Número de Programa:
{proceso.numero_programa}", fg_color="transparent")
            label_programa.place(relx=0.3, rely=self.y_axis+0.03,
anchor=ctk.W)

            label_tiempo = ctk.CTkLabel(self, text=f"Operación y
Datos:  {proceso.operacion}", fg_color="transparent")
            label_tiempo.place(relx=0.3, rely=self.y_axis+0.06,
anchor=ctk.W)

            label_resultado_operacion = ctk.CTkLabel(self, text=f"Resultado
de operación: {self.evaluar_operacion(proceso.operacion)}",
fg_color="transparent")
            label_resultado_operacion.place(relx=0.3, rely=self.y_axis+0.09,
anchor=ctk.W)

            self.y_axis += 0.12

    def actualizar_procesos(self, procesos_borrados):
        print(str(procesos_borrados))
        print(str(procesos_borrados.programador))
        self.procesos_terminados.append(procesos_borrados)  # Agregar el
proceso a la lista
        self.renderizar_procesos()

    def evaluar_operacion(self, operacion):
        try:
            resultado = eval(operacion)  # Intenta evaluar la operación
        except Exception as e:
            return False
        return resultado
```

app.py

```
import tkinter as tk
import customtkinter as ctk
from Vistas.inicio import Inicio
from Vistas.captura import Captura
from Vistas.simulacion import Simulacion
import Modelos.lotes as lotes
from Modelos.procesos import Proceso
import time


#Temas y Colores
ctk.set_appearance_mode("dark")  # Modes: system (default), light, dark
ctk.set_default_color_theme("green")  # Themes: blue (default), dark-blue,
green

#Lotes
capacidad = 4
lotes_a = []

#Pantalla de Inicio para obtener el total de procesos
pantallaInicio = Inicio()
pantallaInicio.mainloop()
procesos = pantallaInicio.procesos #3

#Contadores de procesos y lotes
contador_procesos = 0
contador_lotes = 0

#Proceso para capturar Lotes y Procesos
while procesos != contador_procesos:

    #Pantalla de Captura para obtener la captura de procesos
    pantallaCaptura = Captura()
    pantallaCaptura.mainloop()

    captura = pantallaCaptura.captura  #{a,-,1}
    captura["numero_programa"] = contador_procesos + 1  #{programador:
a,operacion: -, tiempo: 1,numeroprograma: 1}
```

```
    #Se obtiene un lote de 4 procesos en lotes cada 4 procesos
    if contador_procesos % capacidad == 0:
        if contador_procesos > 0:
            lotes_a.append(lote)
        #Se genera un lote nuevo para cada 5 procesos o cuando no es el
primer proceso
        lote = lotes.Lote(capacidad)
        contador_lotes = contador_lotes + 1
        lote.no_lote = contador_lotes

    #Se crea el proceso con los datos ya validados de la captura
    proceso = Proceso(captura)
    contador_procesos += 1
    proceso.no_proceso = contador_procesos
    #Se añade el proceso a su lote correspondiente
    lote.procesos.append(proceso)

#Se agrega el lote pendiente a lotes
lotes_a.append(lote)

#Se crea una instancia singleton de lotes con todos lotes
lotes_o = lotes.Lotes(lotes_a)

#Proceso para manejar el tiempo
pantallaSimulacion = Simulacion()
pantallaSimulacion.mainloop()
```

utils.py

```
import tkinter as tk
from Modelos.lotes import Lotes


def validateEntero(numero):
    if numero.isdigit():
        return True
    else:
        tk.messagebox.showerror("ERROR", "Por favor, ingresa un valor entero
válido.")
        return False

def getProcesos():
    lotes = Lotes()._lotes[0]
    if lotes and lotes[0] and lotes[0].procesos:
        return lotes[0].procesos
    return False

def getProcesosPendientes():
    cont = 0
    lotes = Lotes()._lotes[0]
    for lote in lotes:
        for proceso in lote.procesos:
            cont += 1
    return cont-1 if cont>0 else 0

def deleteProcesos(tiempo):
    lotes = Lotes()._lotes[0]
    if lotes and lotes[0] and lotes[0].procesos:
        if lotes[0].procesos[0].tiempo_maximo == tiempo:
            proceso = lotes[0].procesos[0]
            lotes[0].procesos.pop(0)
            return proceso
    return False

def getLotesPendientes():
    lotes = Lotes()._lotes[0]
    return str(len(lotes)-1) if len(lotes)>0 else "0"

def getLotesEjecucion():
    lotes = Lotes()._lotes[0]
    if lotes and lotes[0]:
        return str(lotes[0].no_lote)
    return 'Ninguno'
```

```
def getProcesoEjecucion():
    lotes = Lotes()._lotes[0]
    if lotes and lotes[0] and lotes[0].procesos:
        return str(lotes[0].procesos[0].no_proceso)
    return 'Ninguno'

def getProcesoEjecucionObjeto():
    lotes = Lotes()._lotes[0]
    print("HOLAAAAAAAAAAAAAA")
    if lotes and lotes[0] and lotes[0].procesos:
        print(str(lotes[0].procesos[0].programador))
        return lotes[0].procesos[0]
    if len(lotes) > 1:
        if lotes[1] and lotes[1].procesos:
            return lotes[1].procesos[0]
    print("noooooooooooooooooo")
    return False

def deleteLotes():
    lotes = Lotes()._lotes[0]
    if lotes and lotes[0]:
        if len(lotes[0].procesos) == 0:
            lotes.pop(0)
            return True
    return False
```

Conclusiones

Padilla Perez Jorge Daray:

Para concluir con esta actividad interesante ya que es raro tener que simular que una computadora actual tarde tanto en hacer ciertos procesos y mas cuando es algo tan sencillo, el tema de tener varias ventanas fue todo un reto que el compañero Ernesto soluciono, la comunicación en equipo fue importante para la realización de esta actividad, aprendí juntos a mis compañeros y espero que podamos seguir así de comunicados.

Juan Jesús Sámano Juárez:

Al empezar a realizar esta actividad me costó trabajo entender los requerimientos, con apoyo de mis compañeros pude entender de mejor manera, en un principio realicé un código que cumplía con la mayoría de lo requerido, después con el equipo se decidió reunir 3 trabajos para hacer uno solo, aprendí sobre el manejo de hilos y a trabajar mediante GitHub.

Ernesto Macias Flores:

El desarrollo de un simulador de lotes en Python es una tarea valiosa para comprender los conceptos de planificación de procesos y la ejecución de tareas en un entorno de procesamiento por lotes, como una buena introducción a lo que se viene después, y aunque es una simulación es útil para comparar y despejarnos un poco de lo que es lo normal en la programación.