Sistemas operativos - 3268

Integrantes: Danilo Hernández, Alexis Vega

Fecha: 29/01/2025

Título del proyecto

Desarrollo de un Simulador Simplificado Objetivo general

Diseñar e implementar un simulador que permita a los estudiantes comprender los conceptos clave de gestión de procesos, memoria, concurrencia y entrada/salida a través de un enfoque práctico e interactivo.

Objetivos específicos

1. Implementar la creación, planificación y ejecución de procesos con algoritmos básicos de planificación.

2. Simular la comunicación entre procesos y la gestión de concurrencia con manejo de interbloqueos.

3. Diseñar un modelo básico de gestión de memoria que integre conceptos de memoria virtual y algoritmos de paginación.

4. Simular la gestión de dispositivos de entrada y salida, incluyendo su interacción con procesos y aplicaciones.

Desarrollo

Ahora se explicará porque el algoritmo creado para el desarrollo de esta actividad sigue perfectamente las indicaciones especificadas:

1. Gestión de procesos

- **Creación de procesos**: Cada proceso tiene atributos definidos como ID, prioridad, y tiempo de ejecución.

- **Planificación**: Se implementan los algoritmos FCFS y Round Robin, que pueden seleccionarse dinámicamente desde la interfaz.

- **Estados básicos**: Los procesos cambian entre los estados Nuevo, Listo, Ejecutando, Bloqueado y Terminado, según la lógica de planificación y las acciones realizadas.

2. Concurrencia e interbloqueo

- **Concurrencia**: Se utiliza threading.Lock para garantizar que las operaciones en los procesos sean seguras en entornos concurrentes.
- **Manejo de bloqueos**: Los procesos pueden ser bloqueados de manera controlada y desbloqueados manualmente a través de la interfaz gráfica.
- **Simulación de interbloqueo**: Aunque no hay una detección automática de interbloqueos complejos, el sistema permite gestionar procesos bloqueados.

3. Gestión de memoria

- **Memoria virtual**: La clase Memoria implementa un esquema de memoria virtual con paginación.
- **Algoritmo de reemplazo**: Se utiliza FIFO para reemplazo de páginas.
- **Visualización**: Aunque no hay gráficos avanzados, el estado de la memoria y las fallas se muestran en forma textual.

4. Gestión de dispositivos de E/S

- **Organización básica**: Los procesos bloqueados representan tareas en espera de recursos externos o dispositivos.
- **Simulación de colas**: Los procesos bloqueados se gestionan en una cola separada (cola_bloqueados).
- **Interacción de procesos con dispositivos**: Los procesos bloqueados pueden ser seleccionados y desbloqueados manualmente desde la interfaz.

5. Interfaz de usuario

- **Interfaz gráfica**: La interfaz gráfica (Tkinter) permite agregar procesos, cambiar algoritmos, ejecutar procesos, desbloquear procesos, y borrar todos los procesos.
- **Amigable y funcional**: Las acciones y el estado de los procesos se muestran de forma clara y detallada en el cuadro de salida.

En general, el algoritmo cubre todas las funcionalidades requeridas para simular un sistema operativo simplificado. Si deseas expandir o detallar algún aspecto, indícalo.

Ahora se presentará el código:

```
import threading
from collections import deque
import tkinter as tk
from tkinter import ttk
import random

# Clase Proceso
class Proceso:
```

```
    def __init__(self, id_proceso, prioridad, tiempo_ejecucion, bloqueado=False):
        self.id = id_proceso
        self.prioridad = prioridad
        self.tiempo_ejecucion = tiempo_ejecucion
        self.estado = "Bloqueado" if bloqueado else "Nuevo"  # Estados: Nuevo, Listo, Ejecutando, Bloqueado, Terminado

    def __repr__(self):
        return f"Proceso(ID={self.id}, Prioridad={self.prioridad}, Tiempo={self.tiempo_ejecucion}, Estado={self.estado})"

# Clase Planificador
class Planificador:
    def __init__(self, algoritmo="FCFS", quantum=2):
        self.cola_listos = deque()
        self.cola_bloqueados = deque()
        self.cola_terminados = []
        self.algoritmo = algoritmo
        self.quantum = quantum
        self.lock = threading.Lock()

    def agregar_proceso(self, proceso):
        if proceso.estado == "Bloqueado":
            self.cola_bloqueados.append(proceso)
        else:
            proceso.estado = "Listo"
            self.cola_listos.append(proceso)

    def ejecutar_procesos(self, callback):
        if self.algoritmo == "FCFS":
            self.fcfs(callback)
        elif self.algoritmo == "Round Robin":
            self.round_robin(callback)

    def fcfs(self, callback):
        while self.cola_listos:
            proceso = self.cola_listos.popleft()
            proceso.estado = "Ejecutando"
            callback(f"Ejecutando {proceso}")
            with self.lock:
                proceso.tiempo_ejecucion -= 1
            if proceso.tiempo_ejecucion <= 0:
                proceso.estado = "Terminado"
                self.cola_terminados.append(proceso)
                callback(f"{proceso} ha terminado.")
            else:
                self.cola_listos.append(proceso)

    def round_robin(self, callback):
        while self.cola_listos:
            proceso = self.cola_listos.popleft()
```

```
            proceso.estado = "Ejecutando"
            callback(f"Ejecutando {proceso}")
            tiempo_ejecutado = min(self.quantum, proceso.tiempo_ejecucion)
            with self.lock:
                proceso.tiempo_ejecucion -= tiempo_ejecutado
            if proceso.tiempo_ejecucion > 0:
                proceso.estado = "Listo"
                self.cola_listos.append(proceso)
            else:
                proceso.estado = "Terminado"
                self.cola_terminados.append(proceso)
                callback(f"{proceso} ha terminado.")

    def desbloquear_proceso(self, proceso_id, callback):
        for proceso in list(self.cola_bloqueados):
            if proceso.id == proceso_id:
                self.cola_bloqueados.remove(proceso)
                proceso.estado = "Listo"
                self.cola_listos.append(proceso)
                callback(f"{proceso} ha sido desbloqueado.")
                return
        callback("Proceso no encontrado en la cola de bloqueados.")

    def limpiar_procesos(self):
        self.cola_listos.clear()
        self.cola_bloqueados.clear()
        self.cola_terminados.clear()

# Clase Memoria
class Memoria:
    def __init__(self, tamanio):
        self.tamanio = tamanio
        self.marcos = [None] * tamanio
        self.fallas = 0

    def asignar_pagina(self, proceso_id):
        if None in self.marcos:
            indice = self.marcos.index(None)
            self.marcos[indice] = proceso_id
        else:
            self.marcos.pop(0)  # Algoritmo FIFO
            self.marcos.append(proceso_id)
            self.fallas += 1

    def __repr__(self):
        return f"Memoria: {self.marcos} | Fallas: {self.fallas}"

# Interfaz de usuario
def interfaz_usuario():
    def agregar_proceso():
        try:
            id_proceso = int(entry_id.get())
```

```
            prioridad = int(entry_prioridad.get())
            tiempo_ejecucion = int(entry_tiempo.get())
            bloqueado = bloqueado_var.get() == 1
            proceso  =  Proceso(id_proceso,  prioridad,  tiempo_ejecucion,
bloqueado)
            planificador.agregar_proceso(proceso)
            limpiar_output()
            mostrar_estado()
        except ValueError:
            output_text.insert(tk.END,   "Error:   Verifica   los   datos
ingresados.\n")

    def desbloquear():
        if not planificador.cola_bloqueados:
            output_text.insert(tk.END, "No hay procesos bloqueados.\n")
            return

        def seleccionar_desbloqueo():
            proceso_id = int(entry_desbloquear.get())
            planificador.desbloquear_proceso(proceso_id,  lambda  mensaje:
output_text.insert(tk.END, mensaje + "\n"))
            desbloquear_window.destroy()
            mostrar_estado()

        desbloquear_window = tk.Toplevel(root)
        desbloquear_window.title("Seleccionar Proceso para Desbloquear")
        tk.Label(desbloquear_window, text="Procesos Bloqueados:").pack()
        for proceso in planificador.cola_bloqueados:
            tk.Label(desbloquear_window, text=f"{proceso}").pack()
        tk.Label(desbloquear_window, text="ID del Proceso:").pack()
        entry_desbloquear = tk.Entry(desbloquear_window)
        entry_desbloquear.pack()
        tk.Button(desbloquear_window,                  text="Desbloquear",
command=seleccionar_desbloqueo).pack()

    def limpiar_output():
        output_text.delete(1.0, tk.END)

    def mostrar_estado():
        output_text.insert(tk.END, "Cola de procesos listos:\n")
        for proceso in list(planificador.cola_listos):
            output_text.insert(tk.END, f"{proceso}\n")
        output_text.insert(tk.END, "\nProcesos bloqueados:\n")
        for proceso in list(planificador.cola_bloqueados):
            output_text.insert(tk.END, f"{proceso}\n")
        output_text.insert(tk.END, "\nProcesos terminados:\n")
        for proceso in planificador.cola_terminados:
            output_text.insert(tk.END, f"{proceso}\n")

    def ejecutar_procesos():
        planificador.ejecutar_procesos(lambda                      mensaje:
output_text.insert(tk.END, mensaje + "\n"))
```

```
        limpiar_output()
        mostrar_estado()

    def cambiar_algoritmo(*args):
        planificador.algoritmo = combo_algoritmo.get()

    def borrar_procesos():
        planificador.limpiar_procesos()
        limpiar_output()
        output_text.insert(tk.END,   "Todos   los   procesos   han   sido
eliminados.\n")

    root = tk.Tk()
    root.title("Simulador de Planificación de Procesos")

    # Entradas
    frame_inputs = tk.Frame(root)
    frame_inputs.pack(pady=10)

    tk.Label(frame_inputs, text="ID Proceso").grid(row=0, column=0)
    entry_id = tk.Entry(frame_inputs)
    entry_id.grid(row=0, column=1)

    tk.Label(frame_inputs, text="Prioridad").grid(row=1, column=0)
    entry_prioridad = tk.Entry(frame_inputs)
    entry_prioridad.grid(row=1, column=1)

    tk.Label(frame_inputs,    text="Tiempo    de    Ejecución").grid(row=2,
column=0)
    entry_tiempo = tk.Entry(frame_inputs)
    entry_tiempo.grid(row=2, column=1)

    bloqueado_var = tk.IntVar()
    tk.Checkbutton(frame_inputs,                          text="Bloqueado",
variable=bloqueado_var).grid(row=3, columnspan=2)

    tk.Button(frame_inputs,             text="Agregar             Proceso",
command=agregar_proceso).grid(row=4, columnspan=2, pady=5)

    # Algoritmo de planificación
    tk.Label(root, text="Algoritmo de Planificación:").pack()
    combo_algoritmo = ttk.Combobox(root, values=["FCFS", "Round Robin"])
    combo_algoritmo.current(0)
    combo_algoritmo.pack()
    combo_algoritmo.bind("<<ComboboxSelected>>", cambiar_algoritmo)

    # Salida
    output_text = tk.Text(root, height=20, width=60)
    output_text.pack(pady=10)

    # Botones de control
```

```
    tk.Button(root,                text="Ejecutar                Procesos",
command=ejecutar_procesos).pack(pady=5)
    tk.Button(root,              text="Desbloquear              Procesos",
command=desbloquear).pack(pady=5)
    tk.Button(root,                text="Borrar                Procesos",
command=borrar_procesos).pack(pady=5)

    root.mainloop()

# Prueba inicial
def main():
    global planificador
    planificador = Planificador()
    interfaz_usuario()

if __name__ == "__main__":
    main()
```

Este código simula un planificador de procesos en un sistema operativo, con una interfaz gráfica utilizando tkinter. A continuación, te explico cada parte:

1. Importaciones

- threading: Se usa para manejar hilos de ejecución, permitiendo ejecutar procesos en paralelo si fuera necesario.
- deque de collections: Utilizado para manejar colas (listas que permiten agregar y quitar elementos de manera eficiente desde ambos extremos).
- tkinter y ttk: Paquetes para crear una interfaz gráfica de usuario (GUI). tkinter es para la creación de ventanas y elementos básicos, y ttk para ciertos widgets con un aspecto más moderno.
- random: Aunque no se usa en el código que muestras, normalmente se usaría para generar números aleatorios.

2. Clase Proceso

- **Atributos**:
  - id: Identificador único del proceso.
  - prioridad: Un número que determina la prioridad del proceso.
  - tiempo_ejecucion: Tiempo de ejecución del proceso.
  - estado: Define el estado del proceso (Nuevo, Listo, Ejecutando, Bloqueado, Terminado).
- **Método __repr__**: Devuelve una representación en forma de cadena del proceso para mostrarlo en la interfaz.

3. Clase Planificador

Esta clase gestiona las colas de procesos (Listos, Bloqueados y Terminados) y define los algoritmos de planificación.

- **Atributos**:
  - cola_listos: Cola de procesos listos para ejecutar.
  - cola_bloqueados: Cola de procesos que están bloqueados.
  - cola_terminados: Cola de procesos que ya han terminado.
  - algoritmo: Algoritmo de planificación a usar (FCFS o Round Robin).
  - quantum: Tiempo máximo de ejecución por ciclo para Round Robin.
  - lock: Un bloqueo para garantizar que solo un hilo modifique los datos a la vez (esto es importante en entornos concurrentes).
- **Métodos**:
  - agregar_proceso: Agrega un proceso a la cola correspondiente (listos o bloqueados).
  - ejecutar_procesos: Ejecuta los procesos según el algoritmo de planificación seleccionado.
  - fcfs: Implementación del algoritmo "First-Come, First-Served" (FCFS), donde se ejecutan los procesos en el orden en que llegaron.
  - round_robin: Implementación del algoritmo "Round Robin", donde cada proceso tiene un tiempo de ejecución limitado (quantum).
  - desbloquear_proceso: Desbloquea un proceso de la cola de bloqueados y lo coloca en la cola de listos.
  - limpiar_procesos: Elimina todos los procesos de las colas.

4. Clase Memoria

- **Atributos**:
  - tamanio: Tamaño de la memoria, que indica cuántos procesos puede almacenar.
  - marcos: Lista que representa los marcos de memoria, donde se almacenan los IDs de los procesos.
  - fallas: Cuenta el número de fallas de página (cuando no hay espacio disponible en la memoria).
- **Métodos**:
  - asignar_pagina: Asigna un proceso a un marco de memoria, utilizando un algoritmo FIFO (el primer proceso en entrar es el primero en salir cuando la memoria está llena).
  - __repr__: Representación en forma de cadena de la memoria.

5. Interfaz de Usuario (GUI)

La interfaz gráfica se crea utilizando tkinter y permite interactuar con el simulador de planificación de procesos.

- **Funciones**:

  - agregar_proceso: Permite agregar un proceso ingresando su ID, prioridad, tiempo de ejecución y si está bloqueado o no.

  - desbloquear: Abre una nueva ventana donde el usuario puede seleccionar un proceso bloqueado para desbloquearlo.

  - limpiar_output: Borra el texto de salida en la interfaz.

  - mostrar_estado: Muestra el estado actual de las colas de procesos (Listos, Bloqueados, Terminados).

  - ejecutar_procesos: Ejecuta los procesos utilizando el algoritmo de planificación seleccionado.

  - cambiar_algoritmo: Permite cambiar el algoritmo de planificación entre FCFS y Round Robin.

  - borrar_procesos: Limpia todas las colas de procesos.

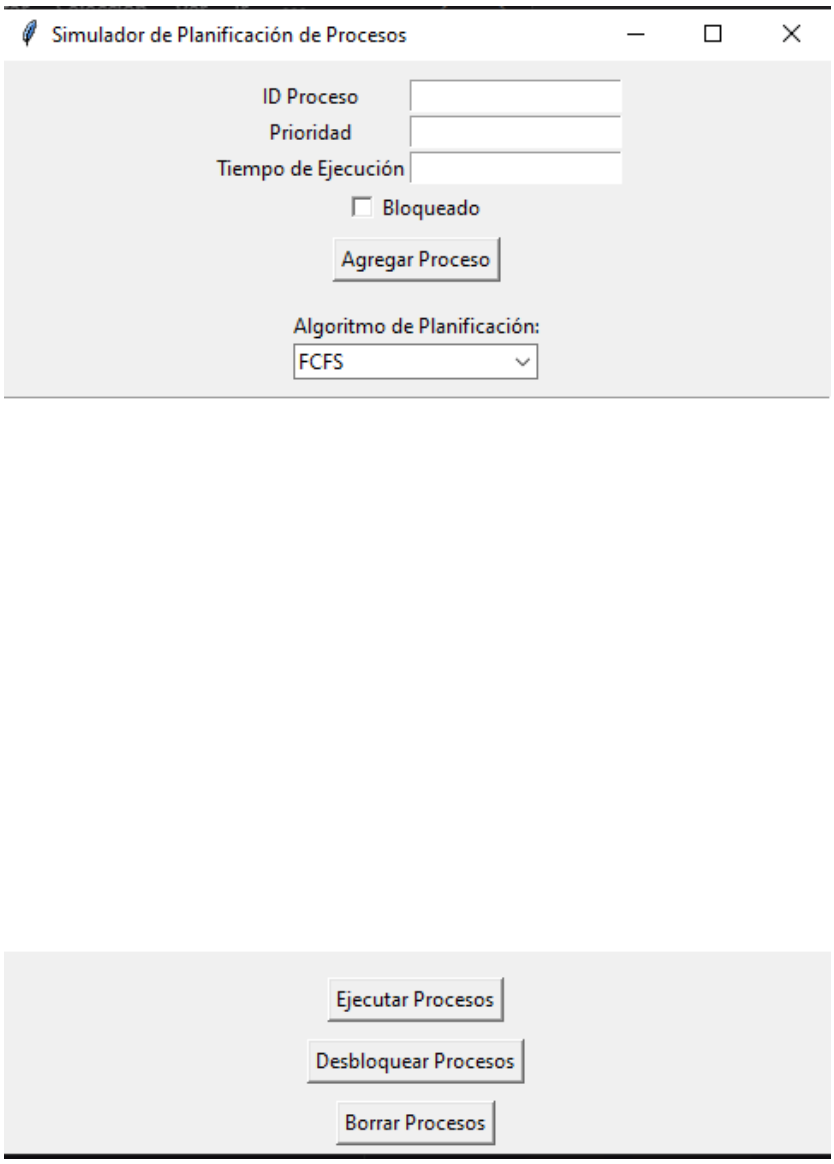
    La ventana principal (root) contiene entradas para agregar procesos, un combo box para elegir el algoritmo de planificación, un área de texto para mostrar el estado y varios botones para ejecutar funciones como agregar procesos, ejecutar procesos, desbloquear procesos y borrar todos los procesos.

    6. Función main

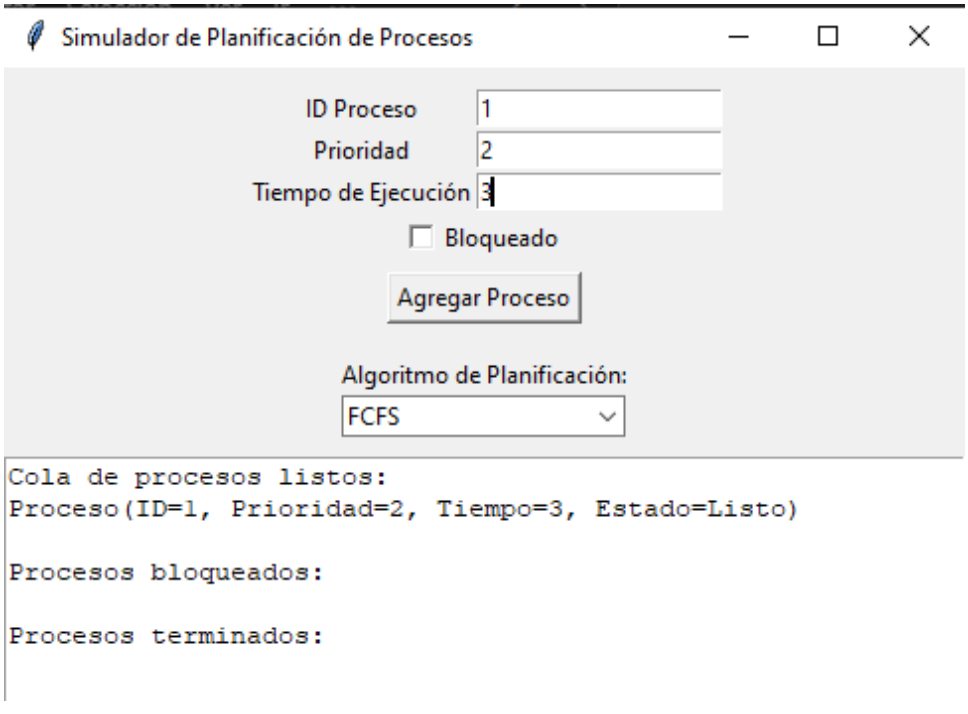
    Esta función crea una instancia de Planificador y ejecuta la interfaz de usuario.

    El código proporciona una simulación básica de un sistema operativo con planificación de procesos, que permite al usuario interactuar con él a través de una interfaz gráfica.

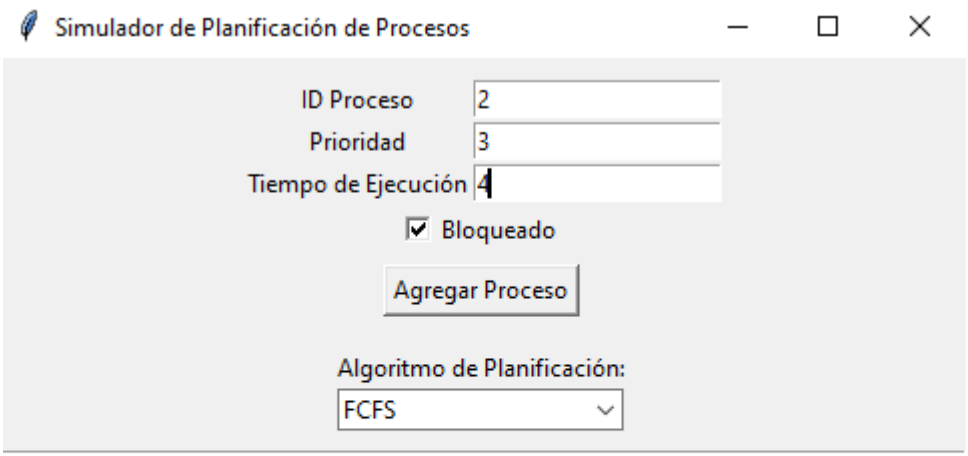
Ejecución



Interfaz de usuario



Ingreso de procesos

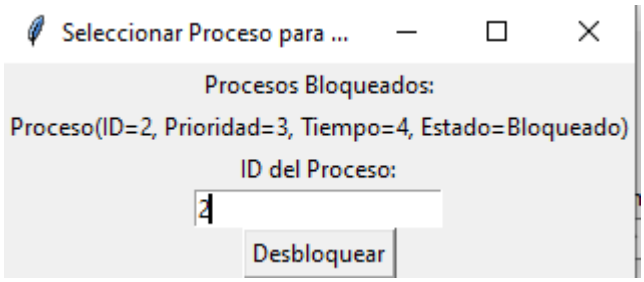



```
Cola de procesos listos:
Proceso(ID=1, Prioridad=2, Tiempo=3, Estado=Listo)

Procesos bloqueados:
Proceso(ID=2, Prioridad=3, Tiempo=4, Estado=Bloqueado)

Procesos terminados:
```

Ingreso de Proceso bloqueado



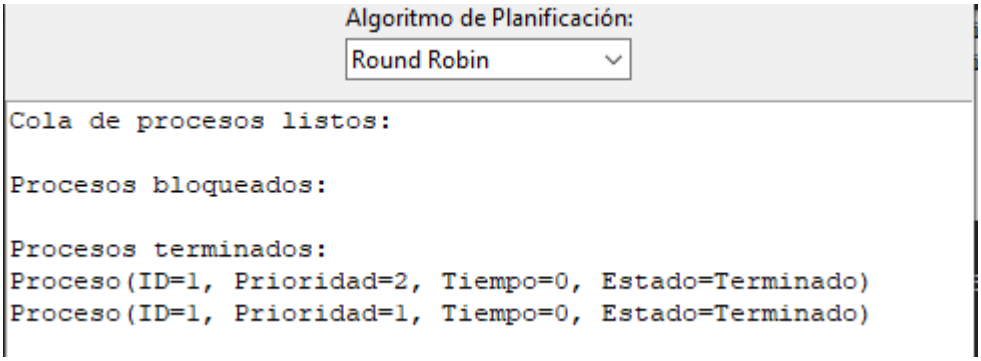
Desbloqueo de procesos por id

```
Cola de procesos listos:

Procesos bloqueados:

Procesos terminados:
Proceso(ID=1, Prioridad=2, Tiempo=0, Estado=Terminado)
Proceso(ID=2, Prioridad=3, Tiempo=0, Estado=Terminado)
```

Ejecución de procesos por FCFS



Ejecución de procesos por Round Robin

Conclusión

- **Simulación de Planificación de Procesos:** El código proporciona una simulación básica pero efectiva de la gestión de procesos en un sistema operativo, utilizando algoritmos de planificación como **FCFS (First-Come, First-Served)** y **Round Robin**. Permite gestionar el ciclo de vida de los procesos (desde su creación hasta su ejecución y finalización), lo que es fundamental en la enseñanza de cómo funcionan los sistemas operativos.

- **Interfaz de Usuario (GUI):** La interfaz gráfica es intuitiva y permite al usuario agregar procesos, cambiar el algoritmo de planificación, ejecutar procesos y ver el estado de las colas de manera clara. Usar tkinter hace que la interfaz sea accesible y fácil de usar, aunque limitada en cuanto a funcionalidades avanzadas.

- **Manejo de Procesos:** Se implementan conceptos clave de los sistemas operativos, como el manejo de procesos bloqueados, listos y terminados. El algoritmo **Round Robin** introduce un control de tiempo (quantum) que refleja una característica común en sistemas operativos modernos.

- **Concurrencia:** El uso de **threading** y de un lock para proteger el acceso a los recursos compartidos en los procesos sugiere que se podría extender el código para simular un entorno concurrente, donde varios procesos puedan ejecutarse en paralelo. Sin embargo, esta parte no se ha implementado completamente en términos de ejecución real en hilos, lo que limita la simulación de procesos concurrentes reales.

- **Memoria Virtual:** El manejo de la memoria utilizando un algoritmo FIFO (First-In, First-Out) para asignar marcos de memoria y contar las fallas de página es un concepto relevante. Aunque se trata de una simulación simplificada, es útil para ilustrar la gestión de memoria en sistemas operativos.

Recomendaciones:

- **Ampliar la Gestión de Concurrencia:** Aunque el código usa `threading` y un `lock`, no se implementa el manejo de procesos en múltiples hilos. Se podría extender para simular la ejecución real de los procesos en diferentes hilos de forma concurrente, agregando más complejidad y realismo a la simulación.

Esto también podría implicar la sincronización de los hilos y el uso de semáforos o barreras.

- **Optimización de la Memoria:** El algoritmo FIFO para la gestión de la memoria es muy simple. Se podría considerar implementar algoritmos más avanzados de sustitución de páginas, como **LRU (Least Recently Used)** o **LRU con página de referencia**, para mejorar la simulación de la gestión de memoria en un sistema operativo real.

- **Interfaz Gráfica Mejorada:** Aunque la interfaz es funcional, podría mejorarse en términos de presentación visual, permitiendo al usuario visualizar los procesos en ejecución de una forma más dinámica, como en una tabla o utilizando gráficos que muestren el tiempo de ejecución o la asignación de memoria. Además, se podrían agregar más detalles sobre los procesos, como su tiempo total de ejecución o el número de ciclos de CPU que han usado.

- **Manejo de Errores:** Se podría mejorar el manejo de errores, especialmente en la interfaz de usuario. Por ejemplo, si se ingresan valores no válidos para la prioridad o el tiempo de ejecución, el sistema debería proporcionar mensajes de error más detallados y específicos.

- **Implementación de Más Algoritmos de Planificación:** Actualmente solo se implementan dos algoritmos de planificación. Sería útil agregar más algoritmos, como **SJF (Shortest Job First)** o **Prioridad**, para dar a los usuarios la opción de experimentar con diferentes técnicas de planificación y observar sus efectos sobre el rendimiento del sistema.

Bibliografía

1. GitHub. (2024). *Simulador de la planificación de recursos y asignación de memoria*. Recuperado de https://github.com/agustinbravop/utn-sims

2. YouTube. (2024). *Algoritmos de planificación de procesos con Python*. Recuperado de https://www.youtube.com/watch?v=QlMv7Pj1pk8

3. GitHub. (2024). *Algoritmos de despacho de procesos*. Recuperado de https://github.com/josefdc/Algoritmos-Despacho

4. UCI. (2024). *Simulador de algoritmos de planificación de procesos*. Recuperado de https://repositorio.uci.cu/jspui/bitstream/ident/TD_1394_08/1/TD_1394_08.pdf

5. Python Software Foundation. (2024). *Documentación oficial de la biblioteca multiprocessing de Python*. Recuperado de https://docs.python.org/es/3.13/library/multiprocessing.html