



**Universidad Nacional Autónoma de Honduras**

**UNAH-Comayagua**

**Asignatura:**

Sistemas Operativo I (IS-412)

**Tema:**

Proyecto Final –

Simulador de Planificación de Procesos

**Docente:**

Ing. Elmer Padilla

**Grupo 10:**

Alumno	NO. Cuenta
Edy Yandel Martínez Tejeda	20221900004
Antonio Josué Vásquez Rojas	20221900346

**Viernes 19 de abril de 2025;**

**Comayagua, Comayagua, Honduras**

## Índice

Introducción.....	3
Objetivos.....	4
Requerimientos del Proyecto .....	5
Desarrollo del programa.....	6
Pruebas realizadas .....	8
Conclusiones.....	9
Anexos .....	10

# Introducción

Este documento explica la creación de un simulador de planificación de procesos, como proyecto final del curso de Sistemas Operativos I. Este simulador se desarrolló en Python y incorpora una interfaz gráfica local mediante la biblioteca Tkinter. El propósito es ilustrar de manera visual y educativa cómo operan diversos algoritmos de planificación de procesos en un sistema operativo.

## Objetivos

- 🚦 Simular algoritmos de planificación de procesos.
- 🚦 Representar de manera visual el diagrama de Gantt.
- 🚦 Calcular y mostrar las principales métricas de rendimiento por proceso.
- 🚦 Facilitar la carga de procesos desde archivo CSV.

## Requerimientos del Proyecto

- 🚦 Simular algoritmos de planificación de procesos.
- 🚦 Representar de manera visual el diagrama de Gantt.
- 🚦 Calcular y mostrar las principales métricas de rendimiento por proceso.
- 🚦 Facilitar la carga de procesos desde archivo CSV.

# Desarrollo del programa

- **Organización del Código**

El simulador se segmenta en diversas partes:

**Clase Proceso:** simboliza cada procedimiento del sistema, con sus características y medidas.

Funciones fcfs, sjf y round\_robin: ponen en práctica el razonamiento de los algoritmos.

**cargar\_csv:** facilita la carga de procedimientos desde un archivo con extensión.csv.

**mostrar\_resultado:** muestra los resultados en una ventana que incluye el gráfico de Gantt y las métricas.

**interfaz:** Crea la interfaz gráfica utilizando Tkinter.

- **Implementación de Algoritmos**

**CFDS:** Ordena los procesos en función del tiempo de entrega. Realiza en la secuencia de llegada.

**SJF:** Elige el procedimiento con el tiempo de ráfaga más corto disponible en cada ciclo.

**Robin Round:** Emplea un cuántum ajustable para realizar procesos de manera rotatoria, utilizando una cola FIFO.

- **Interfaz de Formas Gráficas**

Se estableció una interfaz local mediante Tkinter, la cual facilita:

1. Es preferible elegir el algoritmo de organización.
2. Código del cuántum (si se aplica).
3. Importar un archivo CSV con procedimientos.
4. Visualizar los resultados a continuación:
  - Esquema de Gantt
  - Métricas relacionadas con el proceso (espera, respuesta, regresión)
  - Empleo de la CPU

## Pruebas realizadas

Se realizaron pruebas con distintos archivos CSV conteniendo procesos con diferentes tiempos de llegada y ráfaga, evaluando el comportamiento esperado de cada algoritmo. El simulador responde correctamente, mostrando gráficamente la ejecución y métricas de cada caso.



## Conclusiones

- El proyecto cumple con todos los requerimientos planteados.
- Permite entender claramente las diferencias entre los algoritmos de planificación.
- La interfaz es intuitiva y facilita la carga y visualización de resultados.
- Se podrían añadir mejoras como:
  - Soporte para planificación por prioridades.
  - Ingreso manual de procesos.
  - Exportar resultados a PDF o Excel.

# Anexos

Código fuente:

```
import csv
from collections import deque
from tkinter import *
from tkinter import filedialog, ttk

class Proceso:
    def __init__(self, pid, llegada, rafaga, prioridad=None):
        self.pid = pid
        self.llegada = int(llegada)
        self.rafaga = int(rafaga)
        self.prioridad = int(prioridad) if prioridad else None
        self.rafaga_restante = int(rafaga)
        self.espera = 0
        self.respuesta = None
        self.retorno = 0

#funciones

def fcfs(procesos):
    procesos.sort(key=lambda p: p.llegada)
    tiempo = 0
    gantt = []
    for p in procesos:
        if tiempo < p.llegada:
            tiempo = p.llegada
        p.respuesta = tiempo - p.llegada
        p.espera = p.respuesta
        tiempo += p.rafaga
        p.retorno = tiempo - p.llegada
        gantt.append((p.pid, tiempo - p.rafaga, tiempo))
    return procesos, gantt
```

```
def sjf(procesos):
    tiempo = 0
    completados = 0
    n = len(procesos)
    gantt = []
    lista = []
    while completados < n:
        disponibles = [p for p in procesos if p.llegada <= tiempo and p not in lista]
        if disponibles:
            p = min(disponibles, key=lambda x: x.rafaga)
            lista.append(p)
            p.respuesta = tiempo - p.llegada
            p.espera = p.respuesta
            tiempo += p.rafaga
            p.retorno = tiempo - p.llegada
            gantt.append((p.pid, tiempo - p.rafaga, tiempo))
            completados += 1
        else:
            tiempo += 1
    return procesos, gantt
```

```

def round_robin(procesos, quantum):
    if quantum <= 0:
        raise ValueError("El quantum debe ser mayor que cero")

    if not procesos:
        return [], []

    tiempo = 0
    cola = deque()

    procesos = sorted(procesos, key=lambda x: x.llegada)
    gantt = []
    i = 0
    completados = 0
    n = len(procesos)

    for p in procesos:
        p.rafaga_restante = p.rafaga
        p.respuesta = None

    max_iteraciones = 10000
    iteracion = 0

    while completados < n and iteracion < max_iteraciones:
        iteracion += 1

        while i < n and procesos[i].llegada <= tiempo:
            cola.append(procesos[i])
            i += 1

        if cola:
            actual = cola.popleft()

            if actual.respuesta is None:
                actual.respuesta = tiempo - actual.llegada

            ejecutar = min(quantum, actual.rafaga_restante)
            gantt.append((actual.pid, tiempo, tiempo + ejecutar))
            tiempo += ejecutar
            actual.rafaga_restante -= ejecutar

            while i < n and procesos[i].llegada <= tiempo:
                cola.append(procesos[i])
                i += 1

            if actual.rafaga_restante > 0:
                cola.append(actual)
            else:
                actual.retorno = tiempo - actual.llegada
                actual.espera = actual.retorno - actual.rafaga
                completados += 1
        else:
            tiempo += 1 # CPU idle

    if iteracion >= max_iteraciones:
        raise RuntimeError("El algoritmo excedió el máximo de iteraciones. Posible bucle infinito.")

    return procesos, gantt

```

```

def cargar_csv(path):
    procesos = []
    with open(path, newline='') as archivo:
        lector = csv.DictReader(archivo)
        for fila in lector:
            procesos.append(Proceso(
                pid=fila['ID'],
                llegada=fila['Llegada'],
                rafaga=fila['Rafaga'],
                prioridad=fila.get('Prioridad')
            ))
    return procesos

#Interfaz

def mostrar_resultado(procesos, gantt, ventana):
    resultado = Toplevel(ventana)
    resultado.title("Resultados")

    texto = Text(resultado, width=80, height=20)
    texto.pack()

    texto.insert(END, "\nDiagrama de Gantt:\n")
    for pid, inicio, fin in gantt:
        texto.insert(END, f"[{inicio}-{fin}] {pid} ")

    texto.insert(END, "\n\nMétricas por proceso:\n")
    for p in procesos:
        texto.insert(END, f"Proceso {p.pid}: Espera={p.espera}, Respuesta={p.respuesta}, Retorno={p.retorno}\n")

    uso_cpu = round(sum(p.rafaga for p in procesos) / gantt[-1][2] * 100, 2)
    texto.insert(END, f"\nUso de CPU: {uso_cpu}%\n")

def ejecutar_algoritmo(algoritmo, quantum, ventana):
    path = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if not path:
        return
    procesos = cargar_csv(path)

    if algoritmo == "FCFS":
        result, gantt = fcfs(procesos)
    elif algoritmo == "SJF":
        result, gantt = sjf(procesos)
    elif algoritmo == "Round Robin":
        try:
            q = int(quantum.get())
        except ValueError:
            q = 2
        result, gantt = round_robin(procesos, quantum=q)

    mostrar_resultado(result, gantt, ventana)

def interfaz():
    ventana = Tk()
    ventana.title("Simulador de Planificación de Procesos")
    ventana.geometry("400x250")

    Label(ventana, text="Seleccione algoritmo:").pack(pady=10)
    algoritmo = ttk.Combobox(ventana, values=["FCFS", "SJF", "Round Robin"])
    algoritmo.set("FCFS")
    algoritmo.pack()

    Label(ventana, text="Quantum (solo para Round Robin):").pack()
    quantum = Entry(ventana)
    quantum.insert(0, "2")
    quantum.pack()

```

Archivos de Prueba con los diferentes algoritmos:

