

## Descripción General

Este proyecto es un simulador de gestión de procesos para sistemas operativos que implementa conceptos fundamentales como planificación de procesos, asignación de recursos, comunicación entre procesos mediante memoria compartida y sincronización con mutex. La aplicación cuenta con una interfaz gráfica desarrollada en Tkinter que permite visualizar el comportamiento del sistema en tiempo real.

## Características Principales

- Creación manual de procesos con parámetros aleatorios
- Dos algoritmos de planificación: SJF y Prioridad
- Gestión de recursos (CPU y memoria)
- Demostración del problema Productor-Consumidor
- Terminación normal y forzada de procesos
- Visualización en tiempo real del estado de los procesos

## Arquitectura del Sistema

El sistema está compuesto por los siguientes componentes principales:

### 1. ProcessSchedulerGUI (Capa de Presentación)

Es la interfaz gráfica de usuario desarrollada en Tkinter que actúa como capa de presentación del sistema. Este componente es responsable de:

- **Visualización:** Muestra el estado de todos los procesos, colas, recursos y estadísticas en tiempo real
- **Interacción del usuario:** Captura las acciones del usuario (crear procesos, iniciar/pausar simulación, terminar procesos)
- **Actualización visual:** Refresca los paneles cada 500ms para mostrar cambios en el sistema
- **Coordinación:** Conecta todos los componentes del backend con la presentación visual

Es el punto de entrada principal de la aplicación (main.py) y orquesta la comunicación entre el usuario y el núcleo del sistema operativo simulado.

### 2. Scheduler (Planificador de Procesos)

Es el núcleo de la planificación de procesos, equivalente al scheduler de un sistema operativo real. Sus responsabilidades incluyen:

- **Selección de procesos:** Decide qué proceso debe ejecutarse siguiente según el algoritmo configurado (SJF o Prioridad)
- **Gestión de colas:** Mantiene y organiza las colas de procesos (ready\_queue, waiting\_queue, terminated\_processes)
- **Cambios de estado:** Controla las transiciones entre estados de los procesos (Listo → Ejecutando → Esperando → Terminado)

- **Política de planificación:** Implementa los algoritmos de planificación y gestiona los cambios de contexto
- **Bloqueo/Desbloqueo:** Maneja procesos que esperan recursos o eventos

Es el componente que simula el comportamiento del CPU scheduler de un SO real.

### 3. ResourceManager (Administrador de Recursos)

Es el gestor de recursos del sistema, equivalente al módulo de gestión de memoria y CPU en un SO real. Sus funciones son:

- **Asignación de memoria:** Controla la asignación y liberación de memoria RAM para cada proceso
- **Control de CPU:** Gestiona cuántos procesos pueden ejecutarse simultáneamente según los núcleos disponibles
- **Verificación de disponibilidad:** Valida si hay recursos suficientes antes de crear un proceso
- **Liberación de recursos:** Recupera recursos cuando los procesos terminan
- **Estadísticas de uso:** Calcula y reporta el uso de CPU y memoria en tiempo real

Implementa la política de "first-come, first-served" para la memoria y rechaza procesos cuando no hay recursos suficientes.

### 4. SimulationController (Controlador de Simulación)

Es el director de orquesta que coordina todos los componentes del sistema. Sus responsabilidades incluyen:

- **Control del flujo:** Gestiona el ciclo de vida de la simulación (inicio, pausa, detención)
- **Hilo de ejecución:** Ejecuta la simulación en un thread separado para no bloquear la GUI
- **Velocidad:** Controla la velocidad de ejecución (0.5x a 3.0x)
- **Eventos aleatorios:** Genera eventos estocásticos (bloqueos, desbloqueos) para simular comportamiento realista
- **Integración P-C:** Coordina la ejecución del módulo Productor-Consumidor
- **Callbacks:** Notifica a la GUI cuando debe actualizarse

Actúa como intermediario entre la GUI y los componentes de lógica de negocio.

## 5. ProducerConsumer (Demostración de Sincronización)

Es el módulo de demostración que implementa el problema clásico del Productor-Consumidor. Sus componentes son:

- **Procesos especiales:** Crea dos procesos dedicados (Productor y Consumidor) que interactúan continuamente
- **Coordinación:** Orquesta la interacción entre productor, consumidor, buffer y mutex
- **Demostración educativa:** Muestra visualmente cómo funciona la sincronización entre procesos
- **Estadísticas:** Recopila métricas sobre items producidos, consumidos y estado del buffer

Este módulo es independiente y opcional, diseñado para demostrar conceptos de comunicación y sincronización.

## 6. SharedMemory (Memoria Compartida)

Es el buffer de comunicación entre procesos, simulando memoria compartida en un SO real.

Características:

- **Buffer FIFO:** Cola de tamaño limitado (por defecto 5 items)
- **Operaciones atómicas:** Escritura y lectura controladas
- **Estado verificable:** Permite consultar si está lleno o vacío
- **Registro de accesos:** Mantiene un log de todas las operaciones para análisis
- **Comunicación IPC:** Simula Inter-Process Communication mediante memoria compartida

Es el medio por el cual el productor y el consumidor intercambian información.

## 7. Mutex (Exclusión Mutua)

Es el mecanismo de sincronización que garantiza acceso exclusivo a recursos compartidos.

Características:

- **Lock binario:** Solo un proceso puede poseerlo a la vez
- **Cola de espera:** Procesos bloqueados esperan en orden FIFO
- **Propiedad:** Solo el propietario puede liberar el mutex
- **Prevención de race conditions:** Garantiza que solo un proceso acceda al buffer a la vez
- **Despertar automático:** Cuando se libera, el siguiente proceso en cola lo adquiere automáticamente

Implementa el concepto fundamental de exclusión mutua para evitar condiciones de carrera.

## Flujo de Comunicación

1. **Usuario → GUI:** El usuario interactúa con **ProcessSchedulerGUI**
2. **GUI → Controller:** La **GUI** invoca métodos del **SimulationController**
3. **Controller → Scheduler/ResourceManager:** El controlador coordina el scheduler y el gestor de recursos
4. **Scheduler ↔ ResourceManager:** Se comunican para asignar/liberar recursos según los procesos
5. **Controller → ProducerConsumer:** Si está activo, coordina la ejecución del módulo P-C
6. **ProducerConsumer → SharedMemory/Mutex:** El módulo P-C usa memoria compartida y mutex para sincronización
7. **Todos → GUI:** Todos los componentes reportan su estado a la GUI para visualización

## Módulos Principales

### 1. nucleo\_procesos.py

Contiene las clases fundamentales del sistema de procesos.

#### Clase Process

Representa un proceso individual con sus atributos y estados.

#### Atributos principales:

- pid : Identificador único del proceso
- name: Nombre descriptivo del proceso
- burst\_time: Tiempo total de ráfaga requerido
- remaining\_time: Tiempo restante de ejecución
- priority : Prioridad del proceso (1-10, donde 1 es mayor prioridad)
- memory\_required: Memoria requerida en MB
- state: Estado actual del proceso

#### Estados posibles:

- READY (Listo): El proceso está en la cola de listos
- RUNNING (Ejecutando): El proceso está utilizando la CPU
- WAITING (Esperando): El proceso está bloqueado esperando un recurso
- TERMINATED (Terminado): El proceso ha finalizado

## **Clase Scheduler**

Implementa la planificación de procesos con dos algoritmos.

### **Algoritmos de planificación:**

#### 1. SJF (Shortest Job First)

- Ejecuta primero el proceso con menor tiempo de ráfaga restante
- Minimiza el tiempo de espera promedio
- Puede causar inanición en procesos largos

#### 2. Prioridad

- Ejecuta procesos según su nivel de prioridad (1-10)
- 1 = mayor prioridad, 10 = menor prioridad
- Puede causar inanición en procesos de baja prioridad

Colas gestionadas:

- ready\_queue : Procesos listos para ejecutar
- running\_process: Proceso actualmente en ejecución
- waiting\_queue : Procesos bloqueados
- terminated\_processes: Procesos finalizados

## **Clase SimulationController**

Controla la simulación y coordina los componentes.

Funcionalidades:

- Gestiona el hilo de simulación
- Controla velocidad de ejecución (0.5x a 3.0x)
- Maneja eventos aleatorios (bloqueos, desbloqueos)
- Integra el módulo Productor-Consumidor

## **Clase ProcessGenerator**

Genera procesos con parámetros aleatorios realistas.

Características:

- 45 nombres predefinidos de aplicaciones reales
- Distribución gaussiana para prioridades
- Previene duplicación de nombres
- Genera parámetros dentro de rangos configurables

## **2. administrador\_recursos.py**

### **Clase ResourceManager**

Gestiona los recursos del sistema: CPU y memoria.

Recursos administrados:

- CPU: Número de núcleos disponibles
- Memoria: Capacidad total en MB

Funcionamiento de asignación de memoria:

1. El proceso solicita memoria al crearse
2. Se verifica si hay memoria disponible
3. Si hay suficiente, se asigna y el proceso pasa a cola de listos
4. Si no hay suficiente, el proceso es rechazado
5. Al terminar, la memoria se libera automáticamente

## **3. Comunicacion\_Sincronizacion/**

Este módulo implementa el problema clásico del Productor-Consumidor.

### **Clase SharedMemory (memoria\_compartida.py)**

Implementa un buffer compartido para comunicación entre procesos.

Características:

- Buffer de tamaño configurable (por defecto 5 items)
- Operaciones atómicas de lectura/escritura
- Registro de accesos para estadísticas

### **Clase Mutex (mutex.py)**

Implementa exclusión mutua para sincronización.

Características:

- Garantiza acceso exclusivo al recurso compartido
- Cola de espera FIFO para procesos bloqueados
- Solo el propietario puede liberar el mutex

Funcionamiento:

- Proceso intenta adquirir mutex
- Si está libre, lo adquiere y continúa
- Si está ocupado, el proceso se bloquea y entra a cola de espera
- Cuando el propietario libera, el siguiente en cola lo adquiere automáticamente

## **Clase ProducerConsumer (productor\_consumidor.py)**

Implementa el patrón Productor-Consumidor completo.

Componentes:

- Productor: Genera items y los coloca en el buffer
- Consumidor: Extrae items del buffer y los procesa
- Buffer: Memoria compartida de tamaño limitado
- Mutex: Garantiza exclusión mutua en acceso al buffer

### **Flujo de ejecución:**

#### **Productor:**

1. Adquirir mutex
2. Verificar si buffer tiene espacio
  - Si está lleno: liberar mutex y bloquearse
  - Si hay espacio: producir item
3. Escribir item en buffer
4. Liberar mutex
5. Despertar consumidor si estaba esperando

#### **Consumidor:**

1. Adquirir mutex
2. Verificar si buffer tiene items
  - Si está vacío: liberar mutex y bloquearse
  - Si hay items: continuar
3. Leer item del buffer
4. Liberar mutex
5. Despertar productor si estaba esperando

## **Sincronización:**

- El productor se bloquea cuando el buffer está lleno
- El consumidor se bloquea cuando el buffer está vacío
- Solo un proceso accede al buffer a la vez (exclusión mutua)

## **4. main.py - Interfaz Gráfica**

### **Clase ProcessSchedulerGUI**

Implementa la interfaz gráfica completa usando Tkinter.

#### **Paneles principales:**

##### **Panel de Control**

- Crear Proceso manualmente
- Iniciar/Pausar/Detener simulación
- Terminar proceso forzadamente
- Iniciar/Detener Productor-Consumidor
- Control de velocidad (0.5x - 3.0x)

##### **Panel de Estadísticas**

- Algoritmo de planificación activo
- Tiempo transcurrido
- Contadores de procesos por estado
- Tiempo de espera promedio
- Uso de CPU y memoria
- Log de eventos en tiempo real

##### **Panel de Cola de Procesos**

- Visualización gráfica de todos los procesos
- Código de colores por estado:
  - Verde: Listo
  - Azul: Ejecutando
  - Naranja: Esperando
  - Gris: Terminado
- Información de cada proceso (PID, nombre, prioridad, tiempo restante)
- Clic derecho para terminar proceso

## **Panel Productor-Consumidor**

- Visualización del buffer compartido
- Estado del mutex (libre/bloqueado)
- Identificación del propietario del mutex
- Estadísticas de producción/consumo

## **5. config.py y config.ini**

Archivo config.ini

Archivo de configuración con parámetros del sistema.

### **[Resources]**

```
num_cpus = 4          # Número de CPUs  
total_memory = 4096    # Memoria total en MB
```

### **[Scheduling]**

```
algorithm = SJF      # Algoritmo: SJF o Prioridad
```

### **[Processes]**

```
min_burst_time = 50    # Tiempo mínimo de ráfaga (ms)  
max_burst_time = 500   # Tiempo máximo de ráfaga (ms)  
min_memory = 50        # Memoria mínima requerida (MB)  
max_memory = 300       # Memoria máxima requerida (MB)
```

## **Clase Config**

Carga y valida la configuración del sistema.

### **Validaciones:**

- CPUs > 0
- Memoria total > 0
- Algoritmo válido (SJF o Prioridad)
- Rangos de parámetros coherentes

## **Comunicación y Sincronización**

### **Problema Productor-Consumidor**

El sistema implementa una solución completa al problema clásico:

Escenario:

- Un productor genera items continuamente
- Un consumidor procesa items continuamente
- Ambos comparten un buffer de capacidad limitada

Restricciones:

- El productor no puede agregar items si el buffer está lleno
- El consumidor no puede extraer items si el buffer está vacío
- Solo un proceso puede acceder al buffer a la vez (exclusión mutua)

Propiedades garantizadas:

- Exclusión mutua: Solo un proceso accede al buffer a la vez
- Progreso: Si un proceso puede avanzar, eventualmente lo hará
- Espera acotada: Los procesos no esperan indefinidamente
- Sin deadlock: No hay situaciones de bloqueo permanente

## **Terminación de Procesos**

### **Terminación Normal**

Ocurre cuando el proceso completa su burst\_time:

1. El contador remaining\_time llega a 0
2. El estado cambia a TERMINATED
3. Se calculan estadísticas (tiempo de espera, turnaround, etc.)
4. Los recursos se liberan automáticamente
5. El proceso se mueve a terminated\_processes

## **Terminación Forzada (Abrupta)**

El usuario puede terminar procesos manualmente:

Métodos disponibles:

1. Clic derecho en proceso: Menú contextual directo
2. Botón "Terminar Proceso": Lista de procesos activos

Impacto:

- Los recursos se liberan instantáneamente
- La memoria queda disponible para nuevos procesos
- Si el proceso tenía un mutex, se libera correctamente
- No afecta la estabilidad del sistema

## **Métricas y Estadísticas**

El sistema calcula y muestra las siguientes métricas:

### **Métricas por Proceso**

- **Tiempo de espera:** Tiempo en cola de listos
- **Tiempo de turnaround:** Tiempo total desde llegada hasta finalización
- **Tiempo de respuesta:** Tiempo desde llegada hasta primera ejecución

### **Métricas Globales**

- **Tiempo de espera promedio:** Media de todos los procesos terminados
- **Turnaround promedio:** Media del tiempo total de procesamiento
- **Uso de CPU:** Porcentaje de núcleos en uso
- **Uso de memoria:** Porcentaje de memoria ocupada
- **Context switches:** Número de cambios de contexto

### **Estadísticas Productor-Consumidor**

- **Items producidos:** Total de items generados
- **Items consumidos:** Total de items procesados
- **Items en buffer:** Contenido actual del buffer
- **Adquisiciones de mutex:** Veces que se adquirió el mutex
- **Bloqueos:** Veces que procesos esperaron el mutex

# Guía de Uso

## Requisitos

- Python 3.7+
- tkinter (incluido en la mayoría de instalaciones de Python)
- Dependencias estándar: `threading, time, random, configparser`

## Uso Básico

1. **Crear Procesos**
  - Click en "Crear Proceso" para agregar procesos manualmente
  - Se generan automáticamente con parámetros aleatorios
  - Aparecerá un diálogo mostrando los detalles del proceso creado
2. **Iniciar Simulación**
  - Click en "Iniciar" para comenzar la ejecución
  - Los procesos se ejecutarán según el algoritmo seleccionado
  - Use "Pausar" para pausar/reanudar
  - Use "Detener" para detener completamente
3. **Ajustar Velocidad**
  - Use el slider de velocidad (0.5x - 3.0x)
  - Mayor velocidad = simulación más rápida
4. **Terminar Procesos**
  - Clic derecho sobre un proceso → "Terminar Proceso Forzadamente"
  - O use el botón "Terminar Proceso" para seleccionar de una lista
5. **Demostración Productor-Consumidor**
  - Click en "Iniciar Productor-Consumidor"
  - Observe la interacción entre productor y consumidor
  - El panel derecho muestra el buffer y el estado del mutex

## Solución de Problemas

### Problema: Error "tkinter no está disponible"

- **En Ubuntu/Debian**
  - sudo apt-get install python3-tk
- En macOS
  - Debería venir incluido
- En Windows
  - Debería venir incluido con Python