Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de Ciencias Computacionales

Ingeniería en Computación

“Modelado de una Máquina de Estados Finitos para la Gestión de Estados de Procesos en Sistemas Operativos”

Teoría de la Computación

Clave: IL357 **| Sección:** D03 **| NRC:** 210927

Profesor: Mtro. Abelardo Gómez Andrade

Presenta:

José de Jesús Ruesga Hernández, 222966138

José Luis Haro Díaz, 219416828

Contenido

Tabla de Ilustraciones

Planteamiento del problema

En sistemas operativos, los procesos adquieren diferentes estados como parte de su ciclo de vida (Nuevo, Listo, Ejecución, Bloqueado, Terminado, etc.) según diferentes eventos en el sistema operativo como solicitudes de entradas y salidas (E/S), asignación de recursos, interrupciones o su finalización. Comprender las transiciones es fundamental para los estudiantes que cursan la materia de Sistemas Operativos, pero su complejidad puede dificultar la visualización dinámica. Este proyecto busca resolver dicha problemática mediante un modelo didáctico basado en máquinas de estados finitos, que simplifique la presentación de los estados y sus transiciones, facilitando el aprendizaje activo.

Objetivo

Diseñar e implementar una Máquina de Estados Finitos que modele los 7 estados clave de un proceso en un sistema operativo, junto con sus transiciones con la finalidad de:

- Visualizar de manera interactiva el ciclo de vida de un proceso.
- Servir como herramienta educativa en cursos de sistemas operativos.
- Simular casos prácticos mediante el software JFLAP.

Justificación

El proyecto de modelado de una máquina de estados finitos para representar los estados de un proceso en sistemas operativos se justifica en el ámbito pedagógico, práctico y tecnológico. La gestión de estados de procesos es un pilar fundamental en los sistemas operativos, pero su enseñanza suele enfrentar desafíos debido a la naturaleza abstracta de los conceptos, por lo que el proyecto busca simplificar la complejidad mediante un modelo visual e interactivo, fomentar el aprendizaje activo en donde los estudiantes puedan simular escenarios sencillos pero realistas e integrar la teoría con la práctica.

Dentro del ámbito académico, el modelo es escalable permitiendo su integración en plataformas de e-learning o laboratorios virtuales, así como generar adaptaciones para mostrar fenómenos más avanzados como *procesos zombie* o conceptos de concurrencia.

Adicionalmente, el modelo propuesto no solo sirve como herramienta académica, sino que puede tener aplicaciones profesionales en entornos como el entrenamiento técnico, el prototipado de algoritmos y la producción de documentación interactiva.

Marco teórico

Teoría de autómatas finitos

Un autómata finito (AF) es un modelo computacional abstracto que se utiliza para reconocer lenguajes regulares, se conforma de un conjunto finito de estados, un alfabeto de entrada, una función de transición que define cómo cambiar entre estados según los símbolos leídos, un estado inicial y uno o más estados de aceptación (Sipser, 2013; Hopcroft et al., 2006). Puede ser determinista (AFD) o no determinista (AFND), y su función principal es determinar si una cadena de símbolos pertenece o no a un lenguaje dado. Los autómatas finitos son la base de muchas aplicaciones como análisis léxico en compiladores o el diseño de circuitos digitales (Immerman, 1999).

1) ***Autómata Finito Determinista (AFD)***: un AFD se define formalmente como una 5-tupla $M = (Q, \Sigma, \delta, q_0, F)$, en donde: Q es un conjunto finito de estados; Σ es un alfabeto finito de entrada; $\delta: Q \times \Sigma \to Q$ es la función de transición; $q_0 \in Q$ es el estado inicial; y $F \subseteq Q$ es el conjunto de estados de aceptación (Sipser, 2013).
   Cada transición $\delta(q, a) = q'$ indica que, estando en el estado q y leyendo el símbolo a, el autómata pasa al estado q'. Bajo esta definición determinista, para cada estado y símbolo de entrada existe exactamente una transición especificada por δ (Hopcroft et al., 2006).
2) ***Autómata Finito No-Determinista (AFND)***: un AFND se define formalmente como una 5-tupla $M = (Q, \Sigma, \Delta, q_0, F)$ con los mismos componentes que un AFD, a excepción de la función de transición $\Delta: Q \times (\Sigma \cup \{\epsilon\}) \to P(Q)$. Lo que significa que desde un estado q y un símbolo de entrada $a \in \Sigma$ (incluyendo la cadena vacía) el autómata puede moverse a varios estados posibles de forma no determinista, y se permiten transiciones ϵ que cambian de estado sin consumir un símbolo (Sipser, 2013; Hopcroft et al., 2006).

Además de los autómatas finitos que reconocen lenguajes, existen modelos que además generan salidas asociadas a sus transiciones o estados, y se define formalmente como una 6-tupla. Estas se denominan máquinas de estados finitos (FSM), a diferencia de los autómatas clásicos que únicamente aceptan o rechazan cadenas, las máquinas finitas procesan una secuencia de entradas y producen una secuencia de salidas (Immerman, 1999).

1) ***Máquina de Mealy***: una máquina de Mealy se define formalmente como una 6-tupla: $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_0)$, en donde: Q es el conjunto de estados finitos; Σ es el alfabeto de entrada; Γ es el alfabeto de salida; $\delta: Q \times \Sigma \to Q$ es la función de transición de estado; $\lambda: Q \times \Sigma \to \Gamma$ es la función de salida; $q_0 \in Q$ es el estado inicial (Sipser, 2013).

   La característica distintiva de la máquina de Mealy es que la salida $\lambda(q, a)$ depende del estado actual y del símbolo de entrada, lo que permite una respuesta rápida del sistema, ya que la salida puede cambiar tan pronto como se recibe una entrada (Hopcroft et al., 2006).

2) ***Máquina de Moore***: una máquina de Moore se define formalmente como una 6-tupla: $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_0)$, en donde: Q es el conjunto de estados finitos; Σ es el alfabeto de entrada; Γ es el alfabeto de salida; δ es la función de transición de estado; $\lambda: Q \to \Gamma$ es la función de salida; q_0 es el estado inicial (Immerman, 1999).

   En una Máquina de Moore la salida $\lambda(q)$ únicamente depende del estado actual, lo que implica mayor estabilidad durante el tiempo que permanece en un estado determinado. Es común en sistemas digitales sincrónicos y diseño de autómatas en hardware (Sipser, 2013).

Problema por resolver

La gestión de procesos es una de las principales funcionalidades de cualquier sistema operativo. Se entiende como procesos a una instancia de un programa en ejecución que incluye las instrucciones del programa, los datos y los recursos necesarios para su ejecución (McIver McHoes & Flynn, 2018). El administrador o planificador de procesos, se encarga de gestionar todos los procesos del sistema, asegurando que se ejecución sea efectuada de forma eficiente y que los recursos disponibles sean asignados de manera óptima; dentro de las tareas del administrador de procesos se incluyen la creación, suspensión, reanudación y terminación de procesos (Stallings, 2012; Tanenbaum, 2015).

El ciclo de vida de un proceso dentro de un sistema operativo se suele representar mediante un diagrama de estados con cinco posibles pasos, los diferentes en **Ilustración 1** estados son:

1. *Nuevo*: el proceso ha sido creado pero aún no ha sido cargado en la memoria para su ejecución, con la creación del proceso viene también la generación de su bloque de control de proceso (BCP).

2. *Listo*: el proceso ya está en memoria y listo para ser ejecutado por lo que espera a que el CPU esté disponible porque el proceso anterior terminó su ejecución o porque se presentó una interrupción.
3. *En ejecución*: el proceso está utilizando los recursos del CPU y se está ejecutando, en sistemas con varios procesadores, varios procesos pueden estar siendo ejecutados de forma paralela.
4. *Bloqueado*: el proceso está esperando algún evento externo como una operación de E/S para continuar con su ejecución, en los sistemas con multiprogramación esto da lugar a que se inicie la ejecución de otro proceso. Estos procesos están en espera de la liberación de algún recurso que está siendo utilizado por otro proceso.
5. *Terminado*: el proceso ha finalizado su ejecución y está esperando a que el sistema operativo lo elimine, habilitando nuevamente los recursos que utilizó para que puedan ser utilizados por un nuevo proceso.

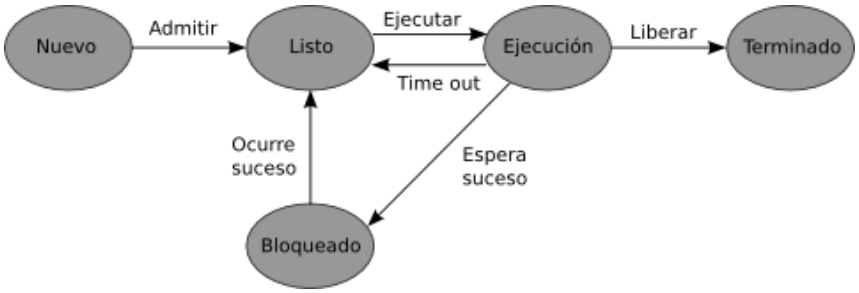


Ilustración 1. Diagrama de 5 estados de la vida de un proceso en un SO.

Adicionalmente, algunos sistemas operativos implementan dos estados más en los que se puede encontrar un proceso cuando se encuentra suspendido y en un periodo de latencia almacenado directamente en disco en **Ilustración 2** (McIver McHoes & Flynn, 2018; Stallings, 2012):

6. *Listo y suspendido*: el proceso está en la memoria secundaria o directamente en disco (zona de intercambio), pero está disponible para su ejecución tan pronto como sea cargado a la memoria principal.
7. *Bloqueado y suspendido*: el proceso está en la zona de intercambio y está esperando algún evento para pasar a bloqueado o más comúnmente a listo y suspendido.

Ilustración 2. Diagrama extendido de 7 estados de la vida de un proceso en un SO.

Metodología

Definición de estados y eventos

El modelo se basa en el esquema de 7 estados para la vida de los procesos en los sistemas operativos modernos, que extiende el modelo clásico de 5 estados al incluir estados suspendidos que generalmente ocurren al no haber disponibilidad de recursos suficientes. Estos estados reflejan escenarios reales como la gestión de memoria limitada o la priorización de procesos.

Se definieron los 7 estados como:

- Nuevo: proceso recién creado (no asignado a la memoria principal).
- Listo: proceso cargado en la memoria (esperando a ser ejecutado).
- Listo y Suspendido: proceso en estado “Listo” pero movido a memoria secundaria o disco por falta de recursos.
- Ejecución: proceso en ejecución activa en la CPU.
- Bloqueado: proceso esperando un evento externo (como Entradas/Salidas).
- Bloqueado Suspendido: un proceso “Bloqueado” movido a memoria secundaria o disco por falta de recursos.
- Terminado: proceso finalizado (ya no consume recursos del sistema).

Se definieron los eventos clave como:

- Admitir: transición de “Nuevo” a “Listo”.
- Activar: reactivación de procesos suspendidos, es decir las transiciones de “Listo y Suspendido” a “Listo”, y de “Bloqueado y Suspendido” a “Bloqueado”.
- Ejecutar: transición de “Listo” a “Ejecución”.

- Suspender: mueve los procesos al disco, es decir las transiciones de “Listo” a “Listo y Suspendido”, y de “Bloqueado” a “Bloqueado y Suspendido”.
- Esperar suceso: existe una solicitud de E/S, transición “Ejecución” a “Bloqueado”.
- Ocurrencia de suceso: se completa la solicitud de E/S, transición de “Bloqueado” a “Listo”.
- Final de tiempo: agotamiento del quantum de CPU (tiempo de ejecución destinado a cada proceso, variable según el algoritmo), transición de “Ejecución” a “Listo”.
- Salir: finalización de un proceso, transición de “Ejecución” a “Terminado”.
- Error: finalización incorrecta de un proceso, transición de “Ejecución a “Terminado”.

Codificación de estados y eventos

En **Tabla 1** y **Tabla 2** se muestran la codificación de los estados para la máquina finita y la codificación de eventos o entradas para la máquina finita, respectivamente.

Tabla 1. Codificación de estados para la máquina finita.

Código	Estado
0	Nuevo
1	Listo
2	Listo y Suspendido
3	Ejecución
4	Bloqueado
5	Bloqueado y Suspendido
6	Terminado

Tabla 2. Codificación de eventos/entradas para la máquina finita.

Código	Evento
0	Admitir
1	Activar
2	Ejecutar
3	Suspender
4	Esperar suceso
5	Ocurrencia de suceso
6	Final de tiempo
7	Salir

Diagrama de transiciones

En **Ilustración 3** se muestra el diagrama de transiciones propuesto para representar el modelo de 7 estados de la vida de un proceso en los sistemas operativos, para conocer el estado consultar la **Tabla 1** y para conocer la transición consultar **Tabla 2**.

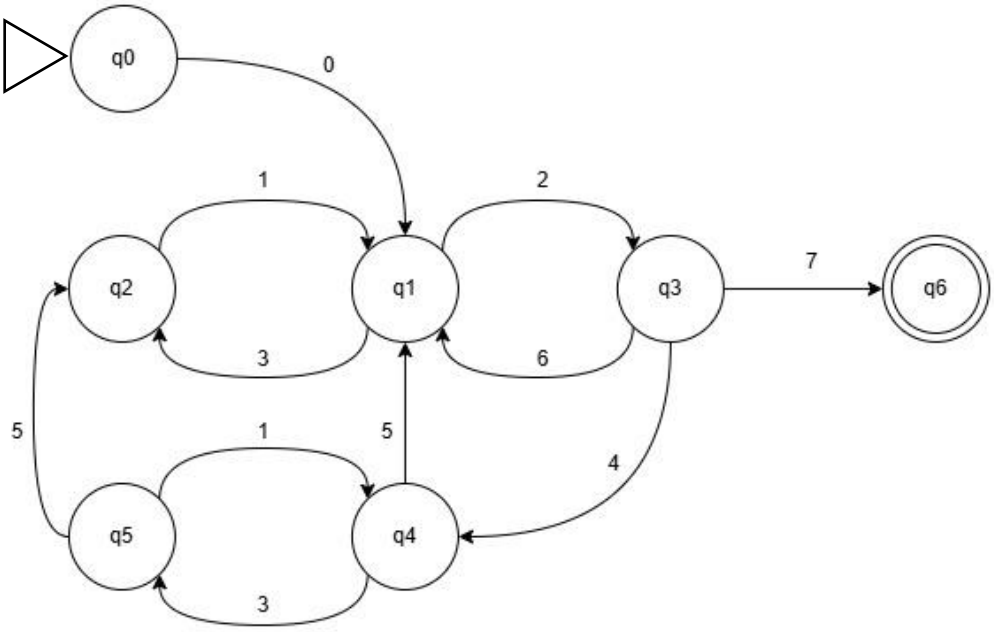


Ilustración 3. Diagrama de transiciones propuesto del modelo de 7 estados.

Tabla de Estado Siguiente (δ)

En **Tabla 3** se encuentra la matriz del estado siguiente (δ), en donde se utiliza -1 para indicar una transición inválida.

Tabla 3. Tabla de Estado Siguiente (δ)

Q \ Σ	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	3	2	-1	-1	-1	-1
2	-1	1	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	4	-1	1	6
4	-1	-1	-1	5	-1	1	-1	-1
5	-1	4	-1	-1	-1	2	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	-1

Tabla de Salidas (g)

En **Tabla 4**, cada celda indica la acción del sistema (salida de texto) asociada a la transición. No existe una salida (N/A) para las transiciones inválidas.

Tabla 4. Tabla de Salidas (g)

Q \ Σ	0	1	2	3	4	5	6	7
0	Proceso admitido	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1	N/A	N/A	Ejecutando proceso	Proceso suspendido	N/A	N/A	N/A	N/A
2	N/A	Reactivar proceso	N/A	N/A	N/A	N/A	N/A	N/A
3	N/A	N/A	N/A	N/A	Solicitud E/S	N/A	Quantum agotado	Proceso terminado
4	N/A	N/A	N/A	Proceso suspendido	N/A	E/S completada	N/A	N/A
5	N/A	Reactivar proceso	N/A	N/A	N/A	E/S completada	N/A	N/A
6	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Simulación en JFLAP

En **Ilustración 4**, se muestra el diagrama de estados junto con sus transiciones y salidas, implementado en JFLAP.

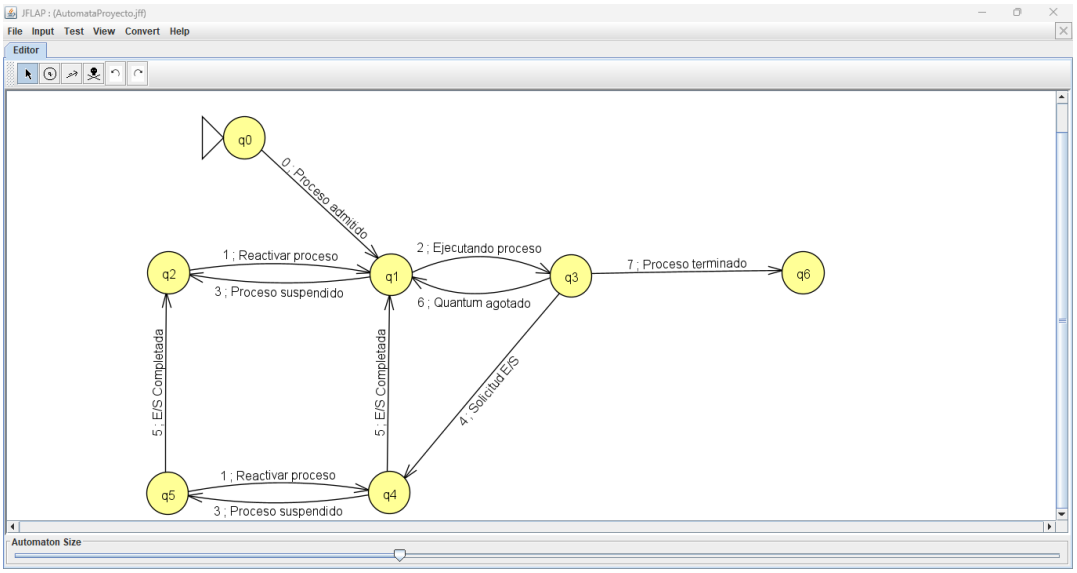


Ilustración 4. Simulación en JFLAP.

Implementación en código

Se realizó una implementación en Python 3.12 del modelo de máquina de estados propuesto para la representación del diagrama de 7 estados de la vida de un proceso en un sistema operativo. Para dicha implementación se desarrolló una interfaz gráfica que permite al usuario visualizar el estado actual y las posibles transiciones, así como los componentes lógicos de la computadora en donde se encuentra un proceso según su estado.

Se implementaron 4 diccionarios, dos son sencillos para la codificación de los eventos y estados respectivamente; y los dos restantes reciben una tupla en su llave para representar las transiciones y las salidas que dependen del estado actual y del evento. El programa cuenta con validaciones para evitar que se seleccione una transición inválida y termina una vez se llega al estado 6 cuando muere el proceso, en **Ilustración 5** se muestra un fragmento de la lógica principal del programa.

```python
def on_event(self, ev: int):
    """Procesa un evento ev (código 0–7)."""
    if ev not in EVENTOS:
        raise ValueError(f"Evento desconocido: {ev}")

    key = (self.estado, ev)
    if key not in TRANSICIONES:
        print(f"Transición inválida: {ESTADOS[self.estado]} + {EVENTOS[ev]}")
        return

    # mensaje de salida
    msg = SALIDAS.get(key)
    # siguiente estado
    siguiente = TRANSICIONES[key]

    # imprimir log
    print(f"Evento  : {ev} – {EVENTOS[ev]}")
    if msg:
        print(f"Acción  : {msg}")
    print(f"{ESTADOS[self.estado]} → {ESTADOS[siguiente]}\n")

    # actualizar estado
    self.estado = siguiente
```

Ilustración 5. Fragmento del código de la lógica principal de transiciones.

Pruebas y resultados

A. *Camino básico hasta terminado*

- Cadena: 027.
- Secuencia: Nuevo → Listo → Ejecutar → Terminado.
- Propósito: Validar el camino más directo desde la creación hasta la finalización de un proceso.

En **Ilustración 6** se encuentra la prueba de la secuencia exitosa en JFLAP en donde se aprecian las diferentes transiciones y el estado final; en **Ilustración 7** se muestra la misma secuencia ejecutada utilizando el programa desarrollado y visualizada desde la línea de comandos; finalmente, en **Ilustración 8** se muestran las transiciones visualizadas desde la interfaz gráfica (GUI) del programa desarrollado.

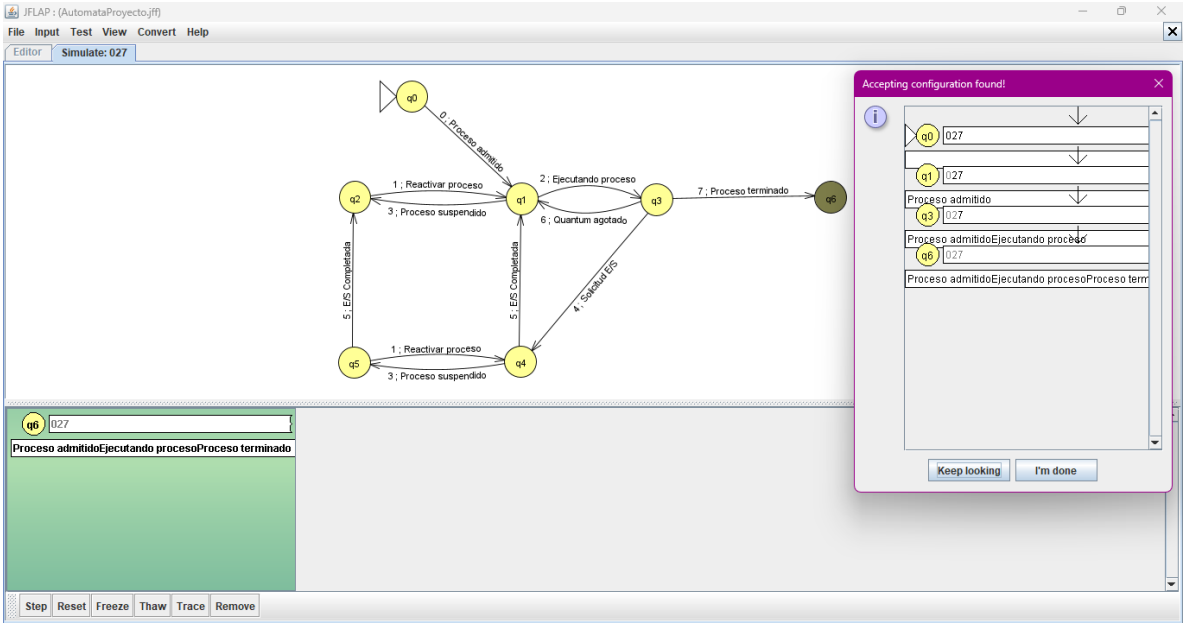


Ilustración 6. Prueba A en JFLAP.

```
PS D:\0_CUCEI\Semestre_6\Teoria_de_la_Computacion\Proyecto> & C:/Users/jrues/miniconda3/python.exe d:/0_CUCEI/Semestre_6/Teoria_de_la_Computacion/Proyecto/source.py
Evento  : 0 – Admitir
Acción  : Proceso admitido
Nuevo → Listo

Evento  : 2 – Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 7 – Salir
Acción  : Proceso terminado
Ejecución → Terminado
```

Ilustración 7. Prueba A en programa desarrollado (interfaz de línea de comandos).

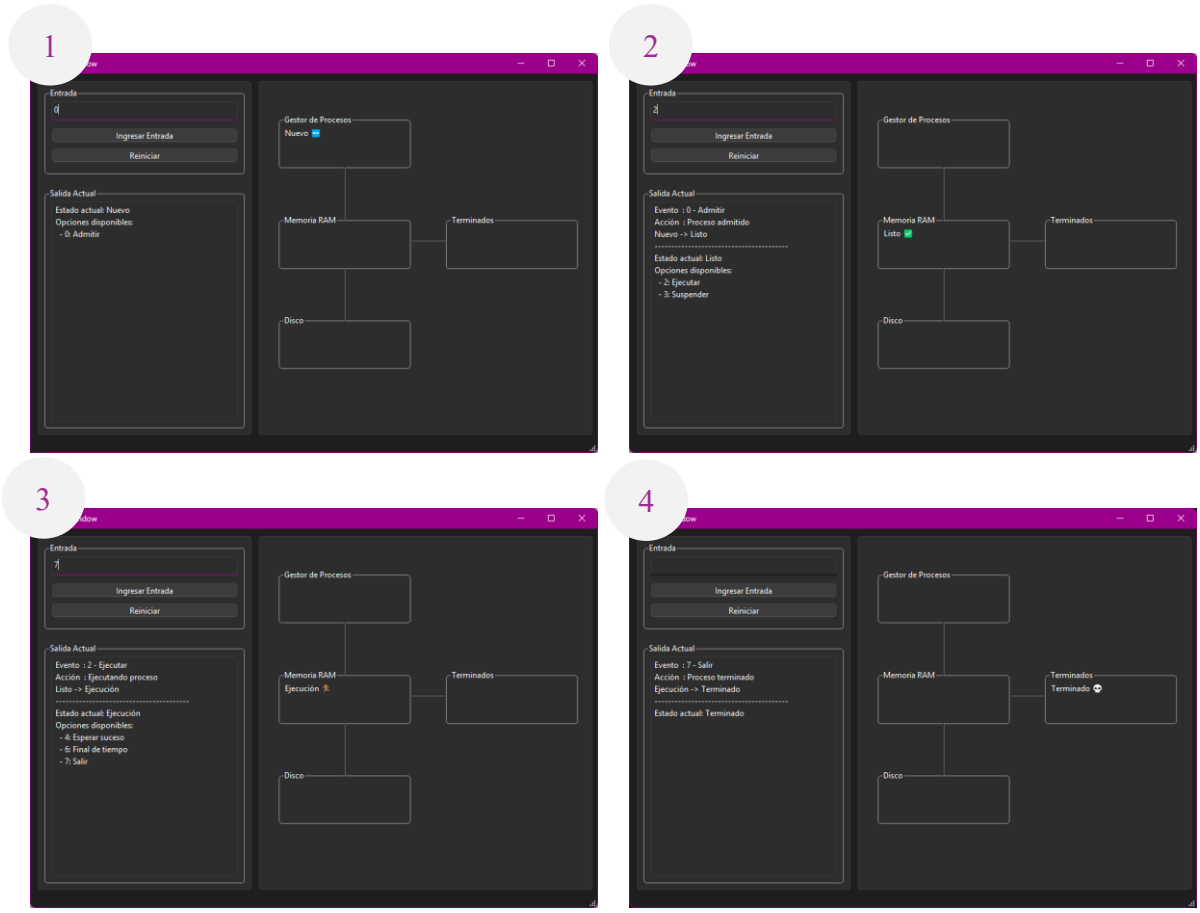


Ilustración 8. Transiciones de prueba A ejecutada desde la GUI del programa.

Por simplicidad, el resto de las únicamente se ejecutaron en la interfaz de comandos, los resultados gráficos pueden encontrarse al final de este documento en **Anexo 1. Resultados Gráficos**.

B. ***Suspensión y reactivación en estado Listo***

- Cadena: 03127.
- Secuencia: Nuevo → Listo → Suspender → Listo y Suspendido → Activar → Listo → Ejecutar → Terminado.
- Propósito: Verificar la correcta suspensión desde Listo y la posterior reactivación.

En **Ilustración 9** se encuentra la prueba de la secuencia exitosa en JFLAP en donde se aprecian las diferentes transiciones y el estado final; en **Ilustración 10** se muestra la misma secuencia ejecutada utilizando el programa desarrollado y visualizada desde la línea de comandos.

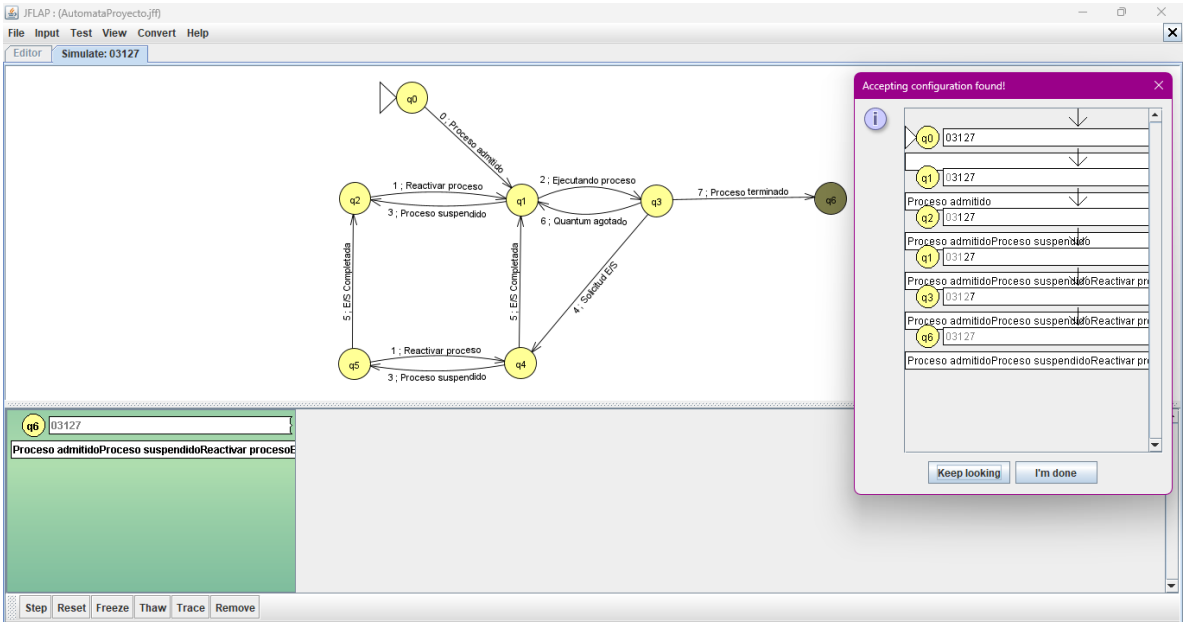


Ilustración 9. Prueba B en JFLAP.

```
PS D:\0_CUCEI\Semestre_6\Teoria_de_la_Computacion\Proyecto> & C:/Users/jrues/miniconda3/python.exe d:/0_CUCEI/Semestre_6/Teoria_de_la_Computacion/Proyecto/source.py
Evento  : 0 – Admitir
Acción  : Proceso admitido
Nuevo → Listo

Evento  : 3 – Suspender
Acción  : Proceso suspendido
Listo → Listo y Suspendido

Evento  : 1 – Activar
Acción  : Reactivar proceso
Listo y Suspendido → Listo

Evento  : 2 – Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 7 – Salir
Acción  : Proceso terminado
Ejecución → Terminado
```

Ilustración 10. Prueba B en programa desarrollado (interfaz de línea de comandos).

C. *Solicitud E/S y terminación normal*

- Cadena: 024527.
- Secuencia: Nuevo → Listo → Ejecutar → Bloqueado → E/S completada → Listo → Ejecutar → Terminado.
- Propósito: Validar el ciclo de E/S completo y la capacidad de retomar ejecución tras estar bloqueado.

En **Ilustración 11** se encuentra la prueba de la secuencia exitosa en JFLAP en donde se aprecian las diferentes transiciones y el estado final; en **Ilustración 12** se muestra la misma secuencia ejecutada utilizando el programa desarrollado y visualizada desde la línea de comandos.

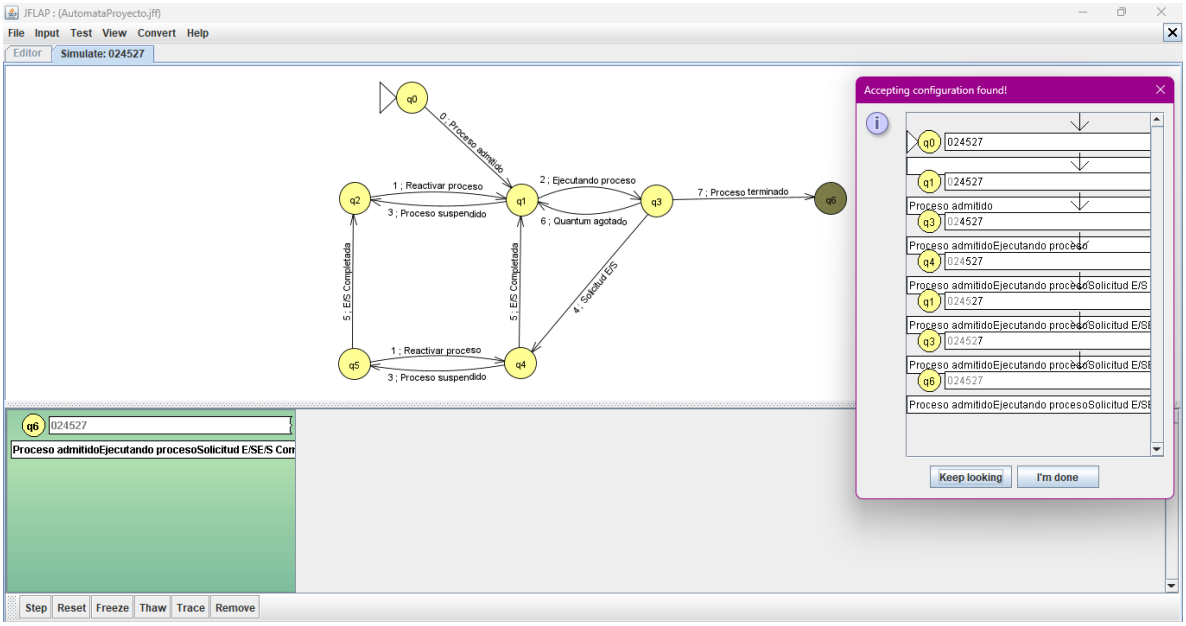


Ilustración 11. Prueba C en JFLAP.

```
PS D:\0_CUCEI\Semestre_6\Teoria_de_la_Computacion\Proyecto> & C:/Users/jrues/miniconda3/python.exe d:/0_CUCEI/Semestre_6/Teoria_de_la_Computacion/Proyecto/source.py
Evento  : 0 – Admitir
Acción  : Proceso admitido
Nuevo → Listo

Evento  : 2 – Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 4 – Esperar suceso
Acción  : Solicitud E/S
Ejecución → Bloqueado

Evento  : 5 – Ocurrencia de suceso
Acción  : E/S completada
Bloqueado → Listo

Evento  : 2 – Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 7 – Salir
Acción  : Proceso terminado
Ejecución → Terminado
```

Ilustración 12. Prueba C en programa desarrollado (interfaz de línea de comandos).

D. *Bloqueado y suspendido con reactivación*

- Cadena: 02431527.
- Secuencia: Nuevo → Listo → Ejecutar → Bloqueado → Bloqueado y Suspendido → Activar → Listo → Ejecutar → Terminado.
- Propósito: Valida la gestión de bloqueos suspendidos y el retorno a ejecución.

En **Ilustración 13** se encuentra la prueba de la secuencia exitosa en JFLAP en donde se aprecian las diferentes transiciones y el estado final; en **Ilustración 14** se muestra la misma secuencia ejecutada utilizando el programa desarrollado y visualizada desde la línea de comandos.

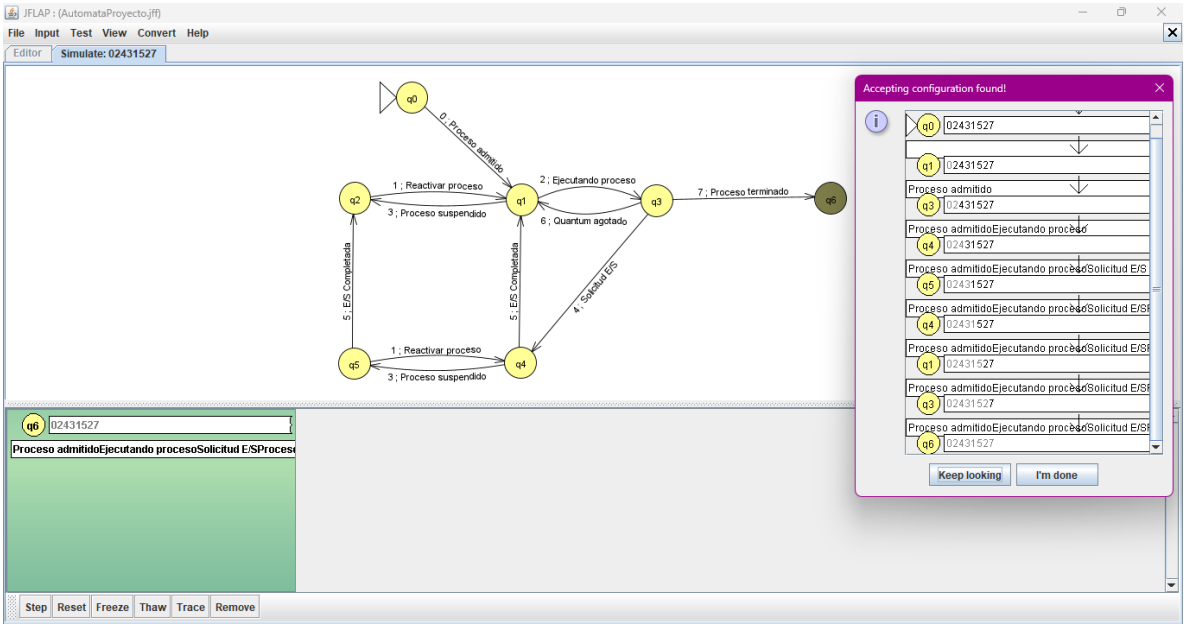


Ilustración 13. Prueba D en JFLAP.

```
PS D:\0_CUCEI\Semestre_6\Teoria_de_la_Computacion\Proyecto> & C:/Users/jrues/miniconda3/python.exe d:/0_CUCEI/Semestre_6/Teoria_de_la_Computacion/Proyecto/source.py
Evento  : 0 – Admitir
Acción  : Proceso admitido
Nuevo → Listo

Evento  : 2 – Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 4 – Esperar suceso
Acción  : Solicitud E/S
Ejecución → Bloqueado

Evento  : 3 – Suspender
Acción  : Proceso suspendido
Bloqueado → Bloqueado y Suspendido

Evento  : 1 – Activar
Acción  : Reactivar proceso
Bloqueado y Suspendido → Bloqueado

Evento  : 5 – Ocurrencia de suceso
Acción  : E/S completada
Bloqueado → Listo

Evento  : 2 – Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 7 – Salir
Acción  : Proceso terminado
Ejecución → Terminado
```

Ilustración 14. Prueba D en programa desarrollado (interfaz de línea de comandos).

E. *Quantum agotado (simulación de multitarea)*

- Cadena: 02624527.
- Secuencia: Nuevo → Listo → Ejecutar → (quantum agotado) → Listo → Ejecutar → Bloqueado → E/S completada → Listo → Ejecutar → Terminado.
- Propósito: Evalúa cómo el sistema maneja el agotamiento de tiempo de CPU y recuperación del control.

En **Ilustración 15** se encuentra la prueba de la secuencia exitosa en JFLAP en donde se aprecian las diferentes transiciones y el estado final; en **Ilustración 16** se muestra la misma secuencia ejecutada utilizando el programa desarrollado y visualizada desde la línea de comandos.

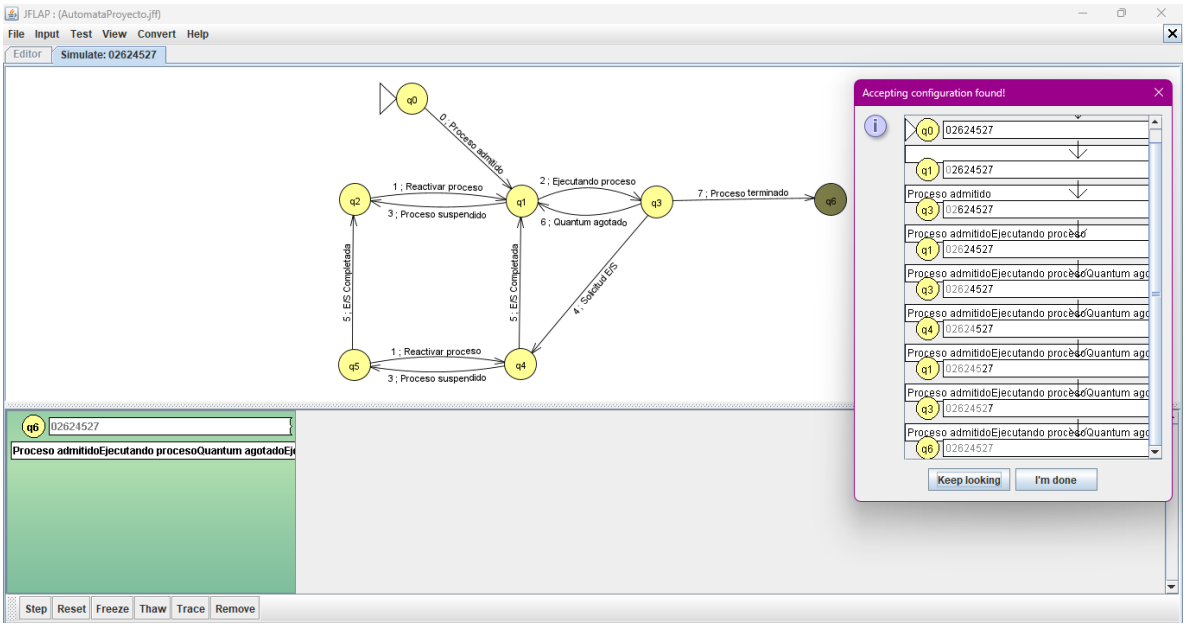


Ilustración 15. Prueba E en JFLAP

```
PS D:\0_CUCEI\Semestre_6\Teoria_de_la_Computacion\Proyecto> & C:/Users/jrues/miniconda3/python.exe d:/0_CUCEI/Semestre_6/Teoria_de_la_Computacion/Proyecto/source.py
Evento  : 0 - Admitir
Acción  : Proceso admitido
Nuevo → Listo

Evento  : 2 - Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 6 - Final de tiempo
Acción  : Quantum agotado
Ejecución → Listo

Evento  : 2 - Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 4 - Esperar suceso
Acción  : Solicitud E/S
Ejecución → Bloqueado

Evento  : 5 - Ocurrencia de suceso
Acción  : E/S completada
Bloqueado → Listo

Evento  : 2 - Ejecutar
Acción  : Ejecutando proceso
Listo → Ejecución

Evento  : 7 - Salir
Acción  : Proceso terminado
Ejecución → Terminado
```

Ilustración 16. Prueba E en programa desarrollado (interfaz de línea de comandos).

F. *Capturas de pantalla adicionales de la GUI del programa*

En **Ilustración 17**, se muestra la interfaz gráfica desarrollada con los diferentes componentes, en rojo se encuentra el panel de entrada en donde el usuario puede ingresar un símbolo y presionar el botón de "Ingresar Entrada" para realizar la transición de estados, en el mismo apartado se encuentra el botón de "Reiniciar" que permite regresar al estado inicial q0. En azul se muestra el panel de salida que muestra información sobre el evento que llevó al estado actual, la acción que corresponde dentro del sistema operativo y la transición realizada; adicionalmente, se muestra el estado actual y la lista de transiciones válidas desde dicho estado. Finalmente, en verde se muestra el panel de representación gráfica en donde se tiene el gestor de procesos (que tiene procesos con estado 'Nuevo'), la memoria RAM (que tiene procesos con estados 'Listo', 'Ejecución', 'Bloqueado'), el disco (que tiene procesos con estados 'Listo y Suspendido', 'Bloqueado y Suspendido'), y terminados que representa la liberación de recursos tras la finalización de un proceso (contiene procesos con el estado 'Terminado').

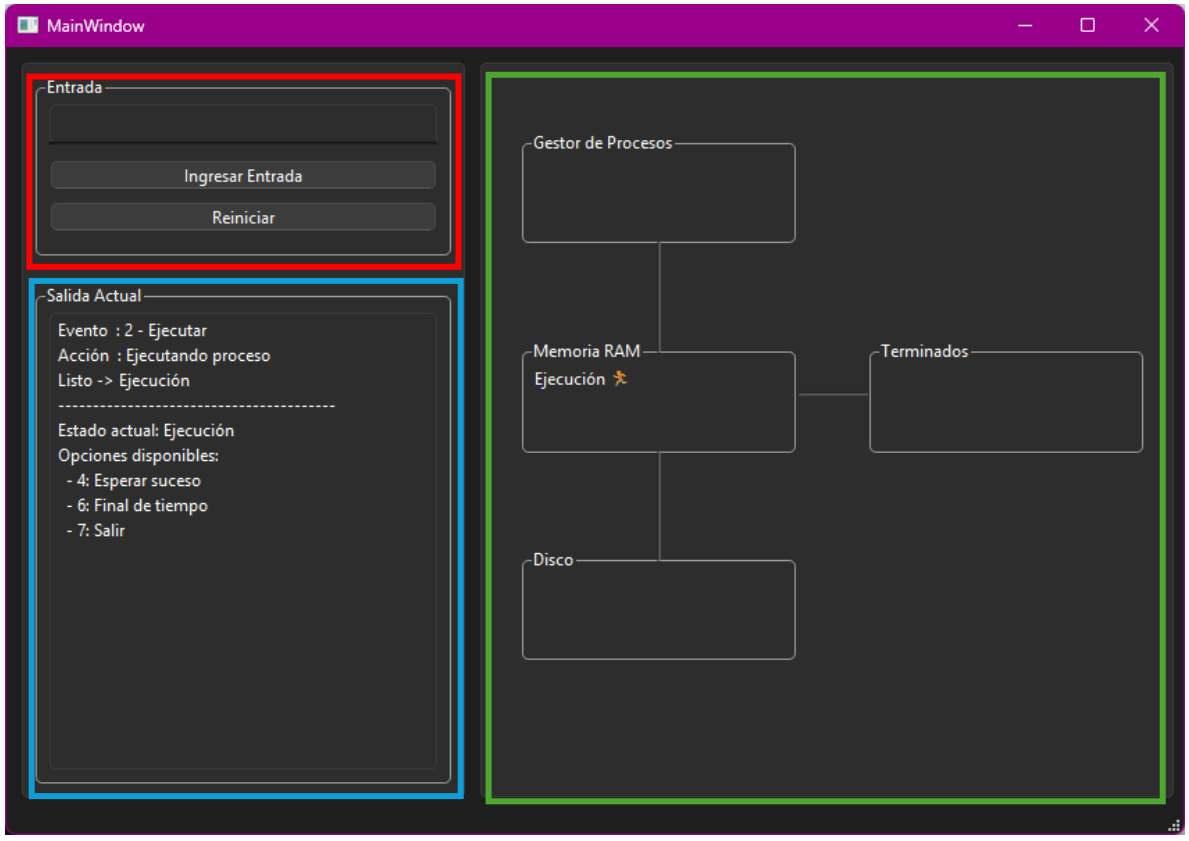


Ilustración 17. Interfaz gráfica de usuario (GUI) del programa desarrollado.

En **Ilustración 18**, se muestra un ejemplo de transición inválida, en donde un estado 0 o 'Nuevo' recibe una entrada '2' que no es válida, ya que el estado 0 únicamente puede recibir una entrada '0' para pasar al estado '1' o 'Listo'. A causa de dicho error, se muestra un error para el usuario en donde se especifica que la transición es inválida.

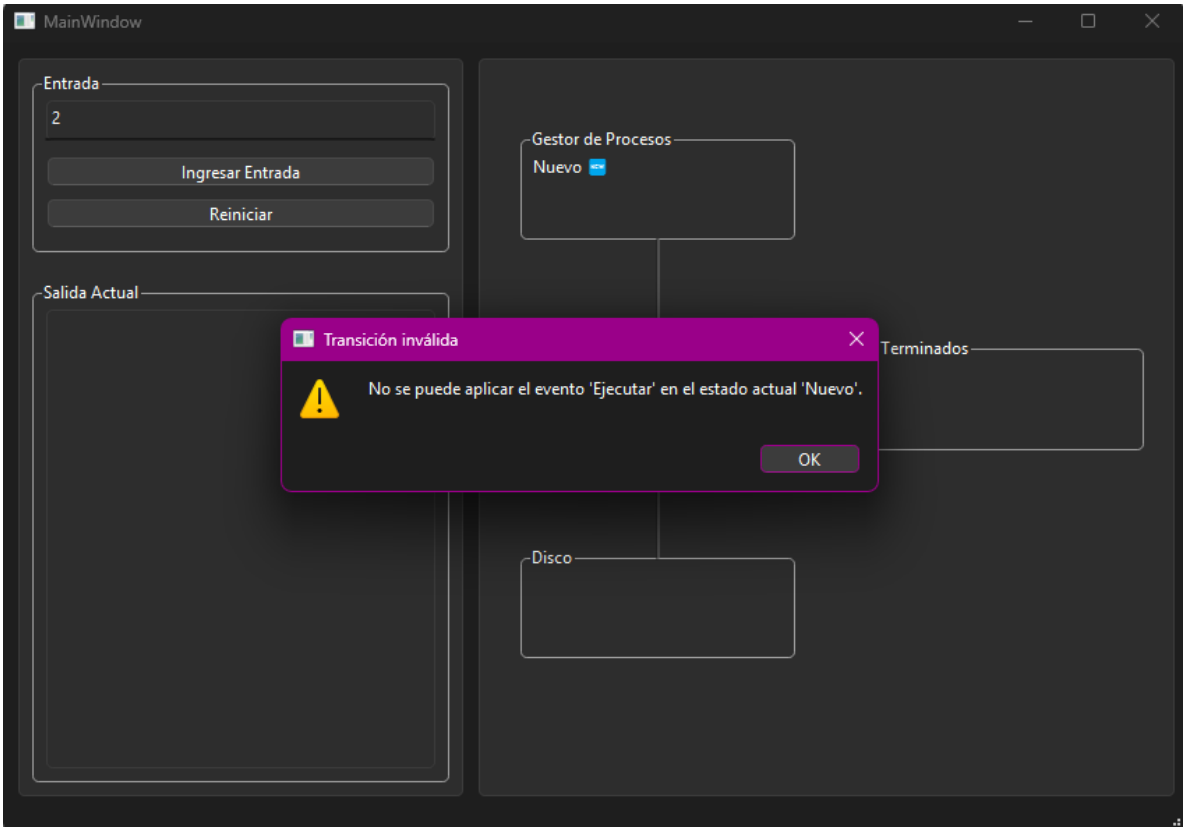


Ilustración 18. Transición inválida.

Conclusiones

El desarrollo de este proyecto permitió modelar de forma clara, estructurada y efectiva el ciclo de vida de un proceso dentro de un sistema operativo moderno, utilizando una máquina de estados finitos que incorporó tanto estados activos como suspendidos. La construcción del autómata, junto con su implementación práctica en JFLAP y en una interfaz gráfica interactiva, demostró la viabilidad del modelo propuesto y su capacidad para replicar fielmente las transiciones entre estados frente a distintos eventos del sistema.

El autómata respondió de manera consistente y eficiente en todas las pruebas realizadas, mostrando un comportamiento óptimo en términos de funcionalidad, estabilidad y fidelidad con el comportamiento real de los procesos. La correcta codificación de la tabla de transiciones y salidas, así como la integración visual de los contenedores para representar los

distintos estados, facilitaron la validación del sistema y reafirmaron su utilidad como herramienta didáctica.

Este enfoque visual e interactivo contribuye significativamente al proceso de enseñanza-aprendizaje en el área de Sistemas Operativos, al ofrecer una representación accesible de conceptos que suelen ser abstractos o difíciles de visualizar. Su diseño modular también permite considerar mejoras futuras, como la incorporación de prioridades de procesos, ejecución concurrente, manejo de interrupciones o incluso su integración en simuladores educativos más complejos.

Referencias

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to automata theory, languages, and computation* (3ra ed.). Pearson.

Immerman, N. (1999). *Computability and complexity*. Prentice Hall.

McIver McHoes, A., & Flynn, I. M. (2018). *Understanding operating systems* (8va ed.). Cengage Learning.

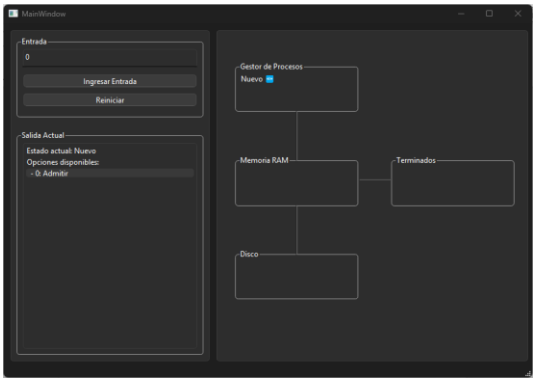
Sipser, M. (2013). *Introduction to the theory of computation* (3ra ed.). Cengage Learning.

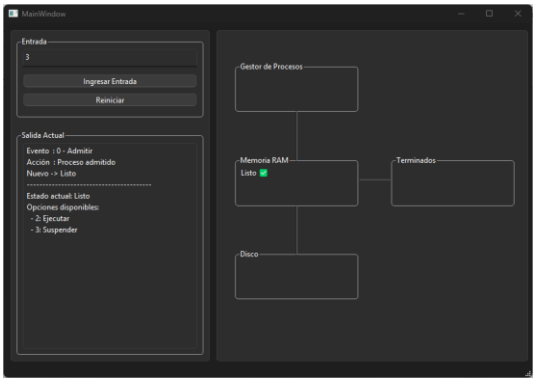
Stallings, W. (2012). *Operating systems: Internals and design principles* (7ma ed.). Prentice Hall.

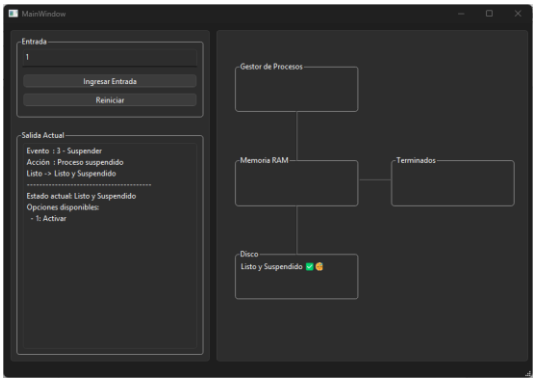
Tanenbaum, A. S. (2015). *Modern operating systems* (4ta ed.). Pearson.

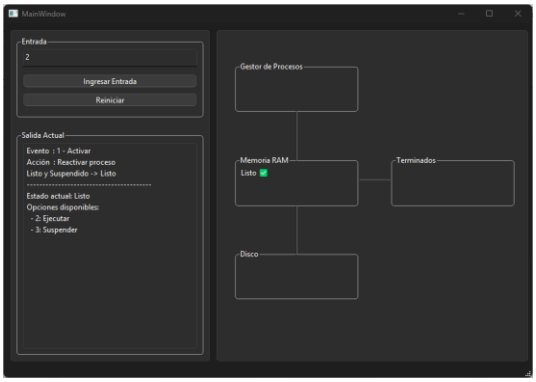
Anexo 1. Resultados Gráficos

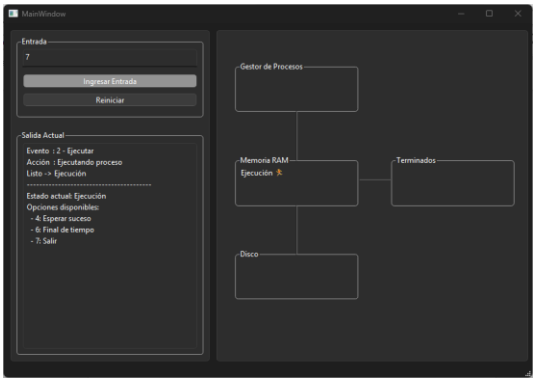
Prueba B: 03127

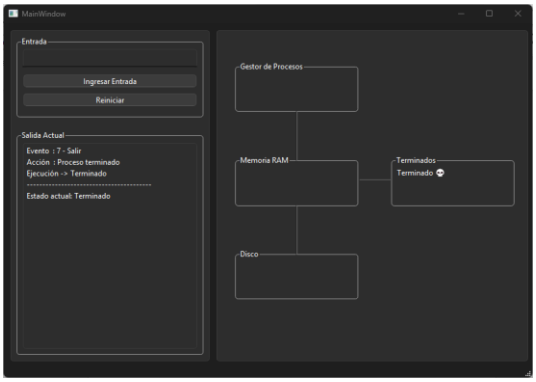




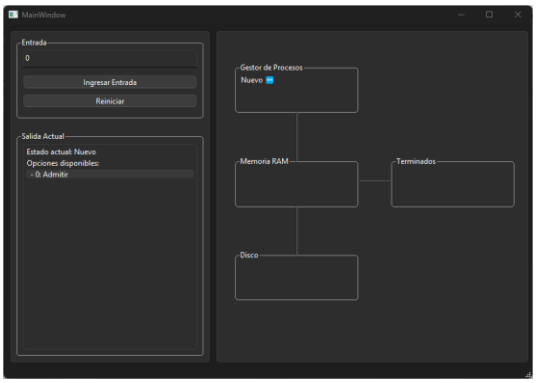


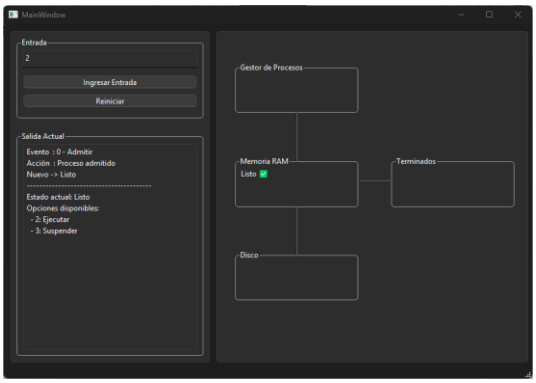


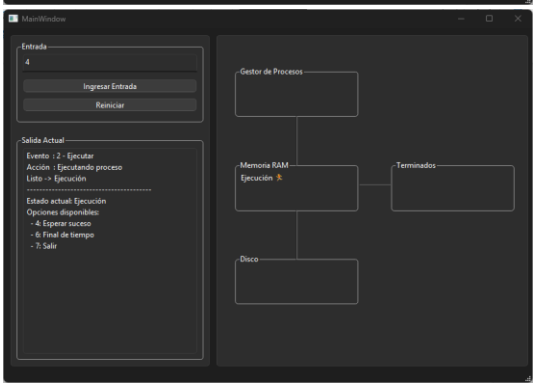


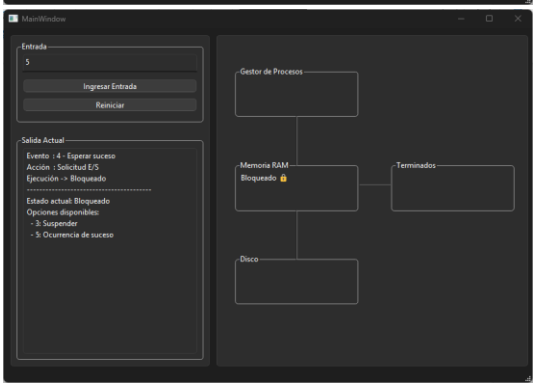


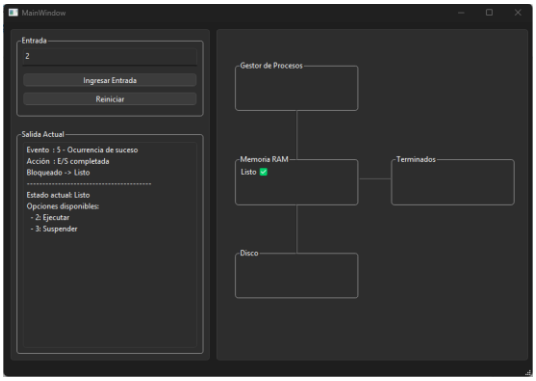
Prueba C: 024527

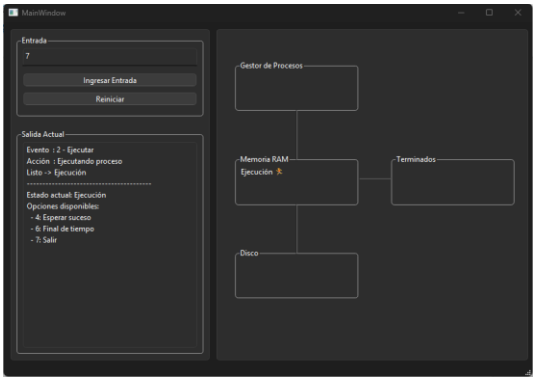


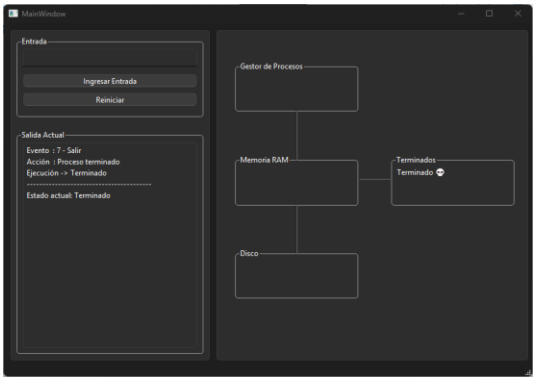




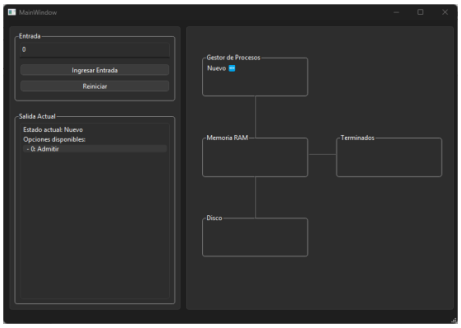


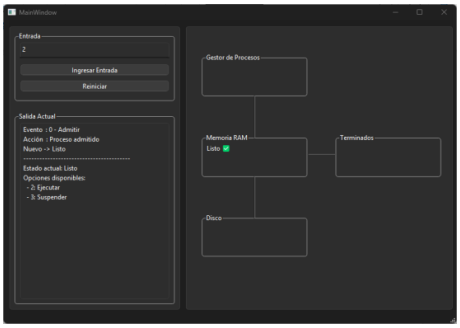


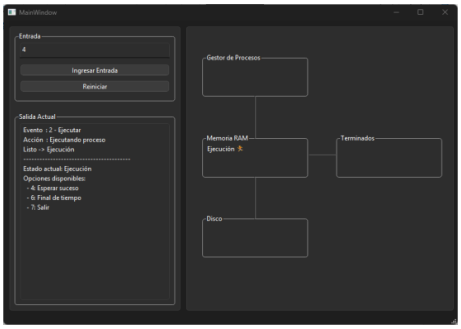


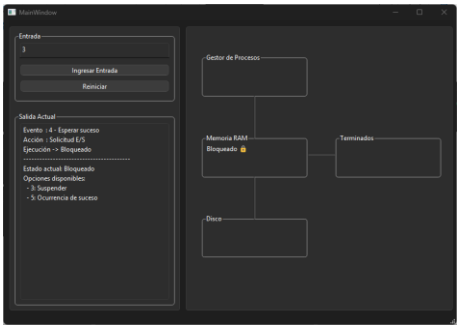


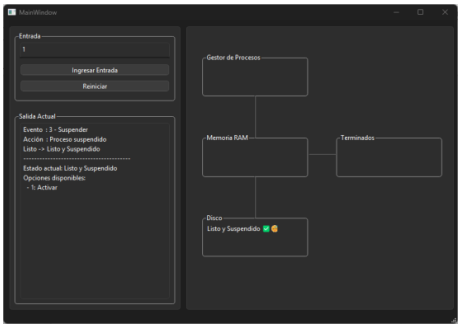
Prueba D: 02431527

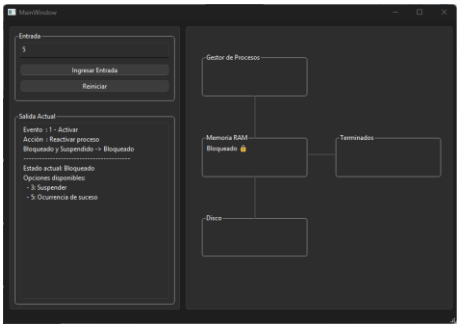


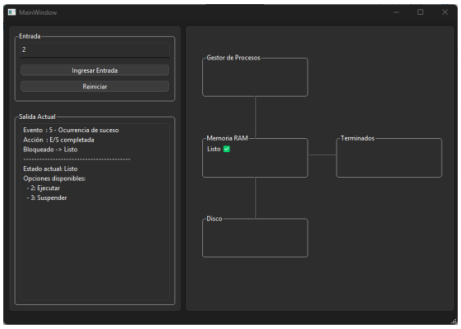


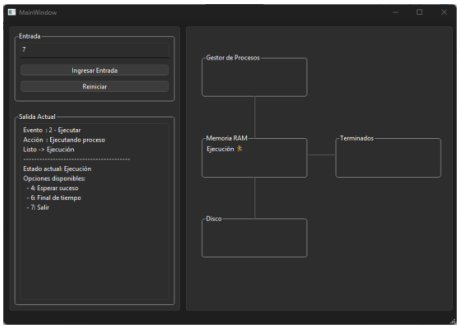


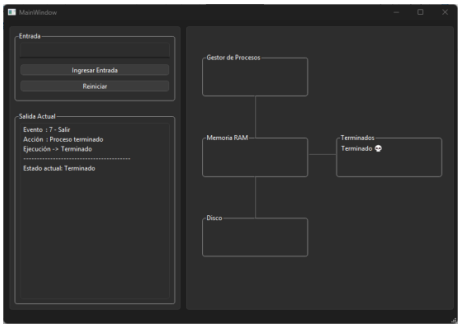












Prueba E: 02624527