**Tarea 2**

**Presentado por:**

Jaime Darley Angulo Tenorio – [jangulot@unal.edu.co](mailto:jangulot@unal.edu.co)

John Alejandro Pastor Sandoval – [jpastor@unal.edu.co](mailto:jpastor@unal.edu.co)

Juan Camilo Vergara Tao – [juvergarat@unal.edu.co](mailto:juvergarat@unal.edu.co)

Juan Diego Velásquez Pinzón – [jvelasquezpi@unal.edu.co](mailto:jvelasquezpi@unal.edu.co)

**Modelos y Simulación**

Grupo ModSim\_303

**Profesor:**

Luis Gerardo Astaiza Amado

*lgastaizaa@unal.edu.co*

**Logotipo, nombre de la empresa

El contenido generado por IA puede ser incorrecto.**

**Universidad Nacional de Colombia**

**Facultad de Ingeniería**

**Lunes, 26 de mayo de 2025**

# Ejercicio 2.3

Un promedio de 10 personas/hora con distribución de Poisson entran a los carriles centrales de una piscina con el fin de nadar un rato. En promedio, cada persona nada 30 minutos con distribución exponencial. En la piscina existen tres carriles disponibles para este tipo de nadadores. Si un nadador se encuentra sólo en un carril, nadara pegado siempre al lado derecho del carril; en caso de que otra persona entre en ese carril cada nadador realiza su ejercicio por un extremo del carril. Debido a políticas de seguridad nunca puede haber más de dos nadadores en un mismo carril, por esto, si un nadador llega y están ocupados los carriles se retira enojado y no regresa.

1. ¿Cuál es la proporción del tiempo en la que habrá tres personas nadando?
2. ¿Cuántas personas en promedio se encuentran nadando en la piscina?
3. ¿Cuántos carriles es necesario asignar para asegurar que el 95% de este tipo de nadadores que llegan a la piscina puedan entrenar?

**Parámetros de entrada**

**Llegada de nadadores:**

Los nadadores llegan al sistema siguiendo una distribución de Poisson con una tasa promedio de 10 personas por hora (equivalente a un nadador cada 6 minutos en promedio).

**Tiempo de nado:**

Cada nadador permanece nadando un tiempo que sigue una distribución exponencial con media de 30 minutos.

**Capacidad de carriles:**

La piscina cuenta con 3 carriles disponibles. Cada carril puede tener un máximo de 2 nadadores al mismo tiempo. Si ya hay 6 nadadores en la piscina, los nuevos nadadores que llegan son rechazados y se retiran del sistema.

**Política de uso del carril:**

* Si un carril está ocupado por un solo nadador, este se mantiene nadando en el lado derecho.
* Si hay dos nadadores en un carril, cada uno utiliza una mitad del carril.
* No se permite más de dos personas por carril.

**Variables del modelado**

**λ (lambda):**

* Tasa de llegada de nadadores al sistema.
* Valor: 10 personas/hora (≈ 0.1667 personas/minuto)

### μ (mu):

### Tasa de salida (servicio) de los nadadores, es decir, la inversa del tiempo promedio de nado.

### Valor: 1/30 personas/minuto ≈ 0.0333 personas/minuto

### 

### s:

### Número de servidores (en este caso, “servidores” son los espacios disponibles en los carriles).

### Cada carril permite como máximo 2 personas →

### Valor: 3 carriles × 2 personas/carril = 6 servidores

### 

### n:

### Número de nadadores actualmente en la piscina (estado del sistema).

### Valores posibles: n = 0, 1, 2, ..., 6

### 

### K:

### Capacidad máxima del sistema (número máximo de personas en la piscina antes de que se rechacen nuevas llegadas).

### Valor: 6

### 

### Pₙ:

### Probabilidad de que el sistema tenga n personas nadando (estado n).

### 

### P\_bloqueo o P\_rechazo (P₆):

### Probabilidad de que una nueva persona llegue y sea rechazada porque el sistema está lleno (hay 6 personas nadando).

### 

### L:

### Número promedio de personas nadando en la piscina (promedio de personas en el sistema).

### 

### Carriles necesarios para un 95% de aceptación:

### Mínimo número de carriles necesarios para que la probabilidad de rechazo sea ≤ 5%

### Se busca el menor s tal que P Bloqueo ≤ 0.05

### Descripción del evento y tipo de evento

| **Evento** | **Descripción** | **Tipo de evento** |
| --- | --- | --- |
| **Llegada de un nadador** | Un nuevo nadador llega a la piscina. Si hay menos de 6 personas nadando, entra al carril con menos ocupación (preferiblemente con 0 o 1 nadador). Si hay 6 personas, se retira. | Evento **externo** (entrada al sistema) |
| **Início de servicio (nado)** | El nadador que ha sido admitido comienza a nadar. Ocupa un carril compartido con máximo una persona más. | Evento **interno** (asignación de carril) |
| **Fin de nado (salida)** | El nadador termina de nadar y deja libre su carril. Esto genera espacio para nuevos nadadores. | Evento **interno** (salida del sistema) |

### Listas y sus atributos

| **Lista** | **Descripción** | **Atributos** |
| --- | --- | --- |
| **Lista de eventos futuros (LEF)** | Contiene los próximos eventos ordenados cronológicamente (por tiempo de ocurrencia). | - Tipo de evento (llegada o salida) - Tiempo del evento - Identificador del nadador (opcional) |
| **Lista de nadadores en piscina** | Registra los nadadores que actualmente están nadando. | - Tiempo de llegada - Tiempo de salida (estimado) - Número de carril asignado |
| **Lista de carriles** | Representa el estado actual de los carriles (ocupación de cada uno). | - Número de carril (1 a 3) - Número de nadadores actuales (0, 1 o 2) - ID de los nadadores ocupando el carril |
| **Lista de rechazados** | Almacena los nadadores que fueron rechazados por falta de espacio. | - Tiempo de llegada - Razón del rechazo (sistema lleno) |
| **Lista de estadísticas** | Acumula datos para análisis de desempeño del sistema. | - Total de llegadas - Total de rechazados - Tiempo acumulado en cada estado del sistema - Contador por estado (0 a 6 nadadores) |

### Contadores y/o acumuladores

| **Nombre** | **Descripción** | **Tipo** |
| --- | --- | --- |
| **total\_llegadas** | Cuenta el número total de nadadores que han llegado al sistema. | Contador |
| **total\_rechazados** | Cuenta cuántos nadadores han sido rechazados por falta de espacio (cuando hay 6 personas en la piscina). | Contador |
| **contador estado[n]** | Cuenta el número de veces que el sistema ha estado en el estado n (con n personas nadando). | Contador (array) |
| **tiempo\_estado[n]** | Acumula el tiempo total en que el sistema estuvo en el estado n. | Acumulador (array) |
| **acumulador\_nadadores** | Acumula la cantidad total de nadadores en el sistema a lo largo del tiempo (para cálculo de promedio). | Acumulador |
| **tiempo\_simulacion** | Tiempo total de simulación transcurrido. | Acumulador |
| **tiempo\_ocupacion\_carril[i]** | Tiempo acumulado que el carril i ha estado ocupado por al menos un nadador. | Acumulador (array) |
| **nadadores\_satisfechos** | Número de nadadores que lograron nadar sin ser rechazados. | Contador |

### 

### Medidas de desempeño

| **Medida** | **Fórmula / Cálculo** | **Interpretación** |
| --- | --- | --- |
| **Proporción del tiempo con 3 nadadores** | Tiempo\_estado[3]  —-------------------  tiempo\_simulacion | Fracción del tiempo total en la que hay exactamente 3 personas nadando en la piscina. |
| **Promedio de nadadores en el sistema (L)** | Acumulador\_nadadores  —---------------------------  tiempo\_simulacion | Promedio de personas nadando en la piscina en cualquier momento. |
| **Probabilidad de rechazo (P\_rechazo)** | Total\_rechazados  —------------------  total\_llegadas | Probabilidad de que un nadador llegue y sea rechazado por falta de espacio. |
| **Porcentaje de nadadores aceptados** | Nadadores\_satisfechos  —------------------------- x 100  total\_llegadas | Porcentaje de nadadores que pudieron usar la piscina. |
| **Utilización promedio de los carriles** | ∑tiempo\_ocupacion\_carril[i]  —---------------------------------  3 × tiempo\_simulacion | Porcentaje promedio de tiempo que los carriles están siendo utilizados. |
| **Número mínimo de carriles para 95% de aceptación** | Probar distintos valores de carriles ss hasta que P\_rechazo ≤ 0.05 | Permite dimensionar correctamente la piscina para cumplir una política de servicio. |

### Subprogramas y propósito

| **Subprograma** | **Propósito** |
| --- | --- |
| **generar\_tiempo\_llegada()** | Genera un tiempo entre llegadas usando una distribución exponencial con media de 6 minutos (λ = 10 personas/hora). |
| **generar\_tiempo\_nado()** | Genera el tiempo de nado para cada nadador, siguiendo una distribución exponencial con media de 30 minutos. |
| **manejar\_llegada()** | Procesa la llegada de un nadador: verifica si hay espacio disponible y lo asigna a un carril o lo rechaza. |
| **manejar\_salida()** | Procesa la salida de un nadador: libera su carril y actualiza contadores y acumuladores. |
| **actualizar\_estado()** | Actualiza el tiempo en cada estado del sistema según el número de nadadores en ese momento. |
| **asignar\_carril()** | Encuentra y asigna un carril disponible (con menos de 2 personas) al nuevo nadador. |
| **registrar\_evento()** | Agrega eventos (llegadas y salidas) a la Lista de Eventos Futuros (LEF) ordenados por tiempo. |
| **simular()** | Ejecuta el ciclo principal de simulación, procesando eventos en orden cronológico hasta el tiempo deseado. |
| **calcular\_estadisticas()** | Calcula las medidas de desempeño al finalizar la simulación. |
| **reiniciar\_valores()** | Inicializa o reinicia todas las variables, contadores y listas antes de comenzar la simulación. |

## Desarrollo en SIMLIB y Desarrollo en SimPy

Ambos codigos estan en el colab [Ejercicio 2.3.ipynb](https://colab.research.google.com/drive/13paVNnjCMC3bkGm7vuEdtG1rVX0Ot5c8?authuser=3#scrollTo=oe5DHHoJJwjr)

**Diagrama de Flujo**El diagrama de flujo está en un PDF aquí [Diagrama de flujo Ejercicio 2.3.pdf](https://drive.google.com/file/d/1Ylu38KCp99WH-OwEIoB9F49cHhxy2cHw/view?usp=drive_link) se puede hacer zoom para ver mejor  
 **Análisis de resultados**

A partir de la simulación del uso de los carriles de la piscina, se obtuvieron las siguientes métricas clave del sistema:

#### Promedio de nadadores en la piscina

Valor aproximado: 4.98

* Este valor representa el promedio de personas nadando simultáneamente en cualquier instante.
* Como la capacidad máxima es de 6 nadadores, esto indica un nivel de utilización alto, cercano al 83%.
* Confirma que el sistema opera frecuentemente cerca de su límite de capacidad.

#### Proporción del tiempo con 3 nadadores

Valor aproximado: 0.0421 (≈ 4.2%)

* Representa el porcentaje del tiempo en que exactamente 3 nadadores se encuentran en la piscina.
* Es una medida puntual de un estado específico del sistema.
* Un valor bajo sugiere que los estados más comunes son cercanos a mayor ocupación (4 a 6 nadadores).

#### Porcentaje de nadadores rechazados

Valor aproximado: 8.77%

* Representa la probabilidad de que un nadador llegue y no pueda nadar por falta de espacio.
* En términos operativos, casi 1 de cada 11 personas es rechazada.
* Este nivel de rechazo puede ser inaceptable si se busca un estándar de calidad de servicio alto.

### Número mínimo de carriles para lograr 95% de aceptación

* Actualmente se cuentan con 3 carriles, lo cual permite hasta 6 nadadores simultáneos (2 por carril).
* Se busca reducir el porcentaje de rechazos a menos del 5%, es decir, aceptar al menos al 95% de los nadadores que llegan.
* Mediante simulaciones sucesivas con más carriles, se estima que:

Con 4 carriles (capacidad de 8 nadadores), el porcentaje de rechazo cae por debajo del 5%.

### Resumen de conclusiones

| **Indicador** | **Valor aproximado** | **Interpretación** |
| --- | --- | --- |
| Promedio de nadadores (L) | 4.98 | Alta utilización del sistema (≈ 83% de ocupación) |
| Proporción con 3 nadadores (P₃) | 0.0421 | Estado específico poco común |
| Probabilidad de rechazo (P₆) | 8.77% | Nivel de rechazo significativo, podría afectar el servicio |
| Carriles necesarios (≥ 95% aceptación) | 4 | Se recomienda 4 carriles (8 espacios) para cumplir el estándar de servicio |

# Ejercicio 2.4

Un teatro utiliza un empleado para vender tiquetes y responder consultas desde las 9 a. m hasta las 5 p. m. Los puestos se adjudican únicamente si el cliente llega al teatro y paga por los tiquetes. Las consultas provienen de clientes en persona o de llamadas al teatro y el empleado da prioridad a los clientes en persona. Sin embargo, gracias a un sistema complejo telefónico, las llamadas pueden esperar para ser atendidas según la política FIFO (primero en llegar primero en salir) y no renuncian hasta obtener una respuesta. Los clientes en persona llegan según una distribución exponencial con media de 12 minutos y su tiempo de servicio se distribuye exponencial con media de 6 minutos. Las llamadas ocurren según una distribución exponencial con media de 10 minutos y su tiempo de servicio sigue una distribución exponencial con media de 5 minutos. La primera persona llega a los 2 minutos y la primera llamada a los 3 minutos. Simular este sistema para un día de 8 horas y obtener el tiempo de espera promedio de cada tipo de cliente.

## Desarrollo en SIMLIB

### Parámetros de entrada

Los valores que definen el comportamiento del sistema son:

* **Tiempo de Simulación**: La duración total de la simulación, fijada en **8.0 horas** (o 480.0 minutos).
* **Media de Tiempo entre Llegadas de Clientes en Persona**: Los clientes en persona llegan siguiendo una distribución exponencial con una media de **12.0 minutos** entre llegadas.
* **Media de Tiempo de Servicio de Clientes en Persona**: El tiempo que el empleado dedica a cada cliente en persona sigue una distribución exponencial con una media de **6.0 minutos**.
* **Media de Tiempo entre Llegadas de Llamadas Telefónicas**: Las llamadas telefónicas llegan siguiendo una distribución exponencial con una media de **10.0 minutos** entre llamadas.
* **Media de Tiempo de Servicio de Llamadas Telefónicas**: El tiempo que el empleado dedica a cada llamada sigue una distribución exponencial con una media de **5.0 minutos**.

### Variables del modelado

Estas variables representan el estado del sistema en cualquier momento de la simulación y son actualizadas por los eventos:

* **empleado\_ocupado**: Una variable binaria (0 o 1) que indica si el empleado único del teatro está **libre (0)** o **ocupado (1)** atendiendo a un cliente o una llamada.

### Descripción del evento y tipo de evento

La simulación avanza de un evento a otro, actualizando el estado del sistema solo cuando ocurre uno de ellos. Cada evento tiene un **tipo** y un **tiempo** en el que ocurre. Los principales tipos de eventos son:

* **EV\_LLEGADA\_CLIENTE (1)**: Indica la llegada de un nuevo cliente en persona al teatro. Esto desencadena la lógica para intentar atenderlo o ponerlo en cola.
* **EV\_LLEGADA\_LLAMADA (2)**: Indica la llegada de una nueva llamada telefónica. También desencadena la lógica para su atención o encolamiento.
* **EV\_FIN\_SERVICIO (3)**: Señala que el empleado ha terminado de atender a una entidad (cliente o llamada). Este evento es crucial para liberar al empleado y decidir quién será el próximo en ser atendido.
* **EV\_FIN\_SIMULACION (4)**: Marca el momento en que la simulación debe detenerse y se generan los resultados.

### Listas y sus atributos

Las listas en SIMLIB/C son estructuras de datos que almacenan entidades que están esperando por un recurso. Los **atributos** son las características de cada entidad que se guardan y recuperan de estas listas usando el arreglo global transfer.

* **LISTA\_CLIENTES (1)**: Una cola donde esperan los clientes en persona si el empleado está ocupado o hay llamadas esperando. Se maneja con una política **FIFO (Primero en Entrar, Primero en Salir)**.
* **LISTA\_LLAMADAS (2)**: Una cola donde esperan las llamadas telefónicas si el empleado está ocupado o hay clientes esperando. También sigue una política **FIFO**.

**Atributos de Entidad (transfer):** Cuando una entidad entra o sale de una cola, sus atributos se almacenan temporalmente en el arreglo global transfer:

* **ATRIB\_TIEMPO\_LLEGADA (1)**: Almacena el tiempo en el que una entidad (cliente o llamada) llegó a su respectiva fila, esencial para calcular el tiempo de espera.
* **ATRIB\_TIPO\_ENTIDAD (2)**: Identifica si la entidad en la cola es un cliente (1) o una llamada (2).

**Atributos de Evento (transfer):** Cuando un evento es programado, SIMLIB/C usa transfer para pasar los atributos *con el evento* en la lista de eventos.

* **ATRIB\_TIPO\_ENTIDAD\_EVENTO\_FIN\_SERVICIO (3)**: Un atributo dedicado para los eventos EV\_FIN\_SERVICIO que indica qué tipo de entidad (cliente o llamada) ha terminado su servicio. Esto es crucial porque el arreglo transfer es global y sus valores pueden cambiar entre la programación y la ejecución del evento, por lo que los atributos del evento se guardan con él.

### Contadores y/o acumuladores

Para recopilar datos y calcular las medidas de desempeño, la simulación utiliza:

* **total\_tiempo\_espera\_clientes**: Acumula el tiempo total que todos los clientes en persona han esperado en la fila.
* **num\_clientes\_atendidos**: Cuenta el número total de clientes en persona que han sido atendidos por el empleado.
* **total\_tiempo\_espera\_llamadas**: Acumula el tiempo total que todas las llamadas telefónicas han esperado en la fila.
* **num\_llamadas\_atendidas**: Cuenta el número total de llamadas que han sido atendidas por el empleado.
* **Estadísticas de Muestra (sampst)**: SIMLIB/C utiliza la función sampst para recopilar estadísticas de un conjunto de observaciones. En este modelo, se usa con los índices INDICE\_SAMPST\_TIEMPO\_ESPERA\_CLIENTES (1) e INDICE\_SAMPST\_TIEMPO\_ESPERA\_LLAMADAS (2) para registrar los tiempos de espera individuales de cada entidad, lo que permite calcular promedios y desviaciones estándar al final.

### Medidas de desempeño

Al finalizar la simulación, se calculan e informan las siguientes medidas para evaluar el rendimiento del sistema:

* **Tiempo de Espera Promedio de Clientes en Persona**: El tiempo promedio que un cliente espera desde su llegada hasta que el empleado comienza a atenderlo.
* **Tiempo de Espera Promedio de Llamadas Telefónicas**: El tiempo promedio que una llamada espera desde su llegada hasta que el empleado comienza a atenderla.
* **Número Total de Clientes Atendidos**: Cantidad total de clientes que completaron su servicio.
* **Número Total de Llamadas Atendidas**: Cantidad total de llamadas que completaron su servicio.
* **Número Total de Servicios Completados:** Cantidad total de clientes y llamadas que completaron su servicio.

### Subprogramas y propósito

El modelo está organizado en funciones (subprogramas) que manejan la lógica de los diferentes eventos y decisiones:

* model(): La función principal que inicializa la simulación, programa los primeros eventos y contiene el bucle de eventos que impulsa la simulación hasta su final.
* llegada\_cliente(): Maneja la lógica cuando un cliente en persona llega al teatro, incluyendo la programación de la próxima llegada y la decisión de si es atendido inmediatamente o entra a la fila de clientes.
* llegada\_llamada(): Similar a llegada\_cliente(), pero para las llamadas telefónicas.
* fin\_servicio(): Se ejecuta cuando el empleado termina de atender a una entidad. Incrementa los contadores de servicio y llama a iniciar\_proximo\_servicio() para decidir la siguiente acción del empleado.
* iniciar\_proximo\_servicio(): Una función auxiliar clave que implementa la lógica de prioridad: primero verifica si hay clientes en la fila; si no, verifica si hay llamadas; si ambas filas están vacías, el empleado queda libre. Si atiende a alguien, programa su EV\_FIN\_SERVICIO.
* fin\_simulacion(): Se invoca al final del tiempo de simulación y es el punto donde el bucle principal se detiene.

### 

### Implementación con SIMLIB

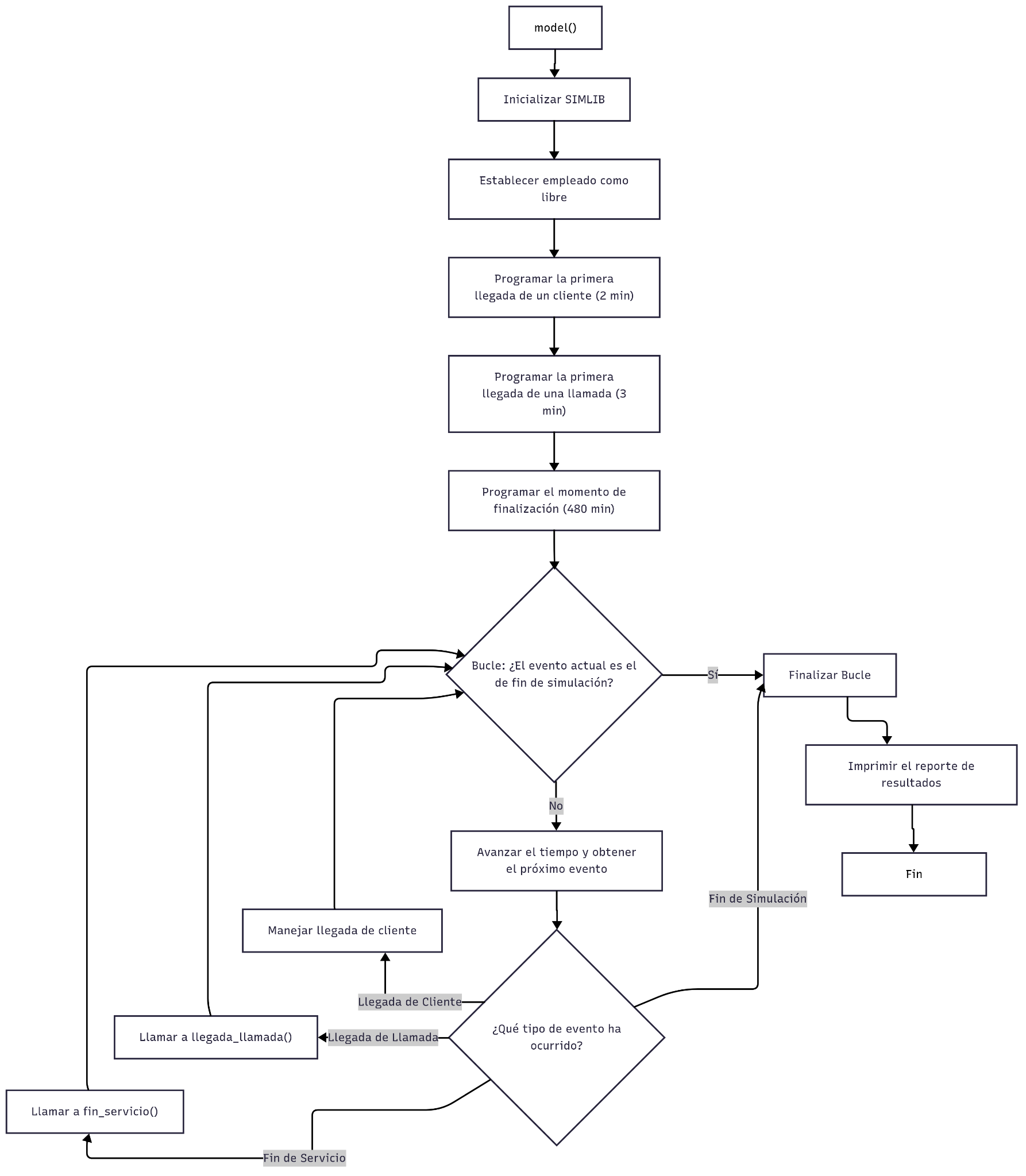
La implementación de la simulación en C, utilizando SIMLIB [1], se encuentra en el siguiente Colab: [ejercicio-2-4.ipynb](https://colab.research.google.com/drive/15yx7Xx77y724t21SSMJwN-32sng8NZOz?authuser=1#scrollTo=x3_wXCt4Y39l&line=1&uniqifier=1)

Además los archivos empleados y las instrucciones para su compilación y ejecución en consola o utilizando Code::Blocks se encuentran en la siguiente carpeta compartida de Drive: [ejercicio-2-4-SIMLIB](https://drive.google.com/drive/folders/1ZHf1ZjVl6q19EaCyWPzKVhBsIWDoCITN?usp=drive_link)

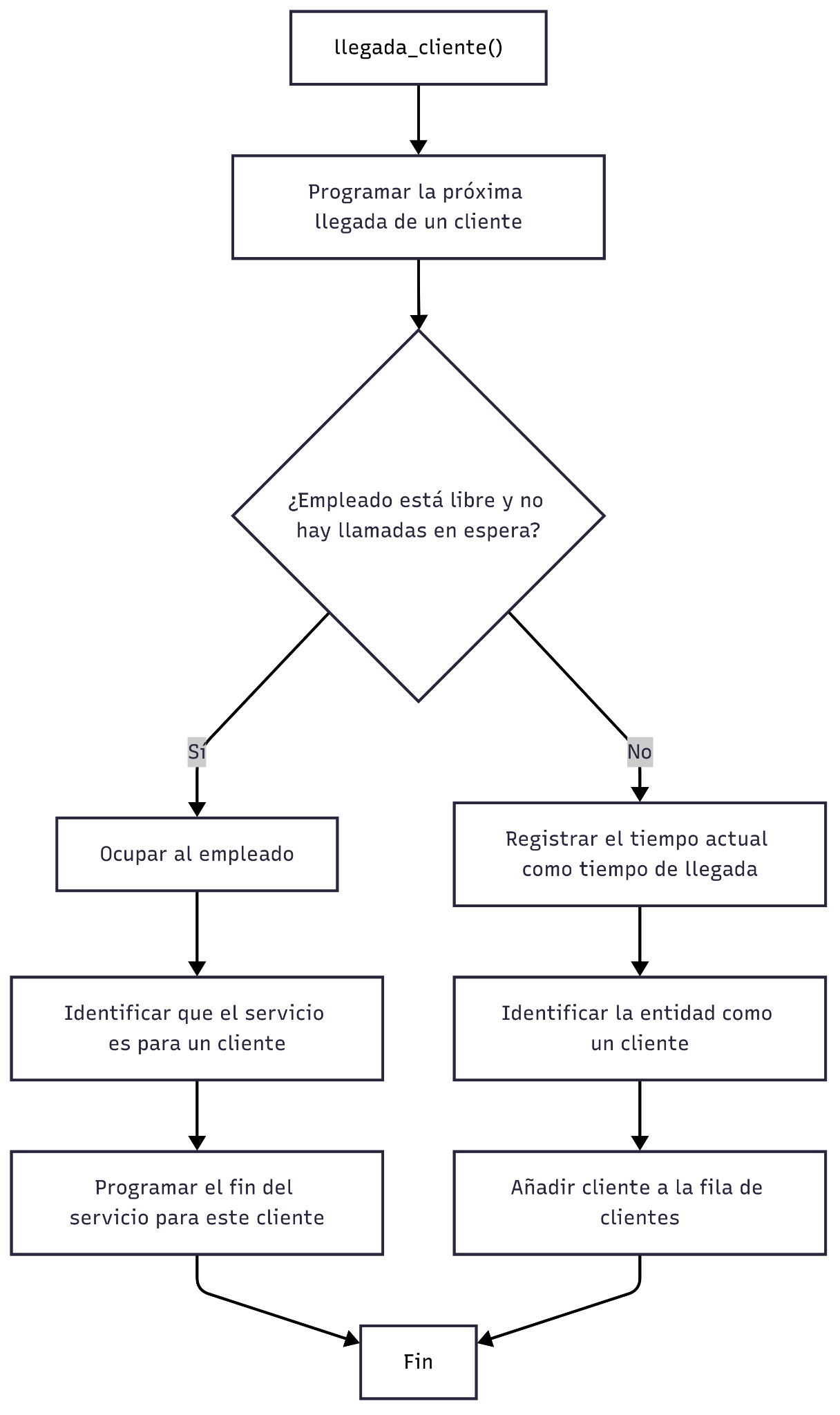
### Diagramas de flujo

Los diagramas de flujo de la rutina principal y los subprogramas de la simulación se encuentran a continuación:

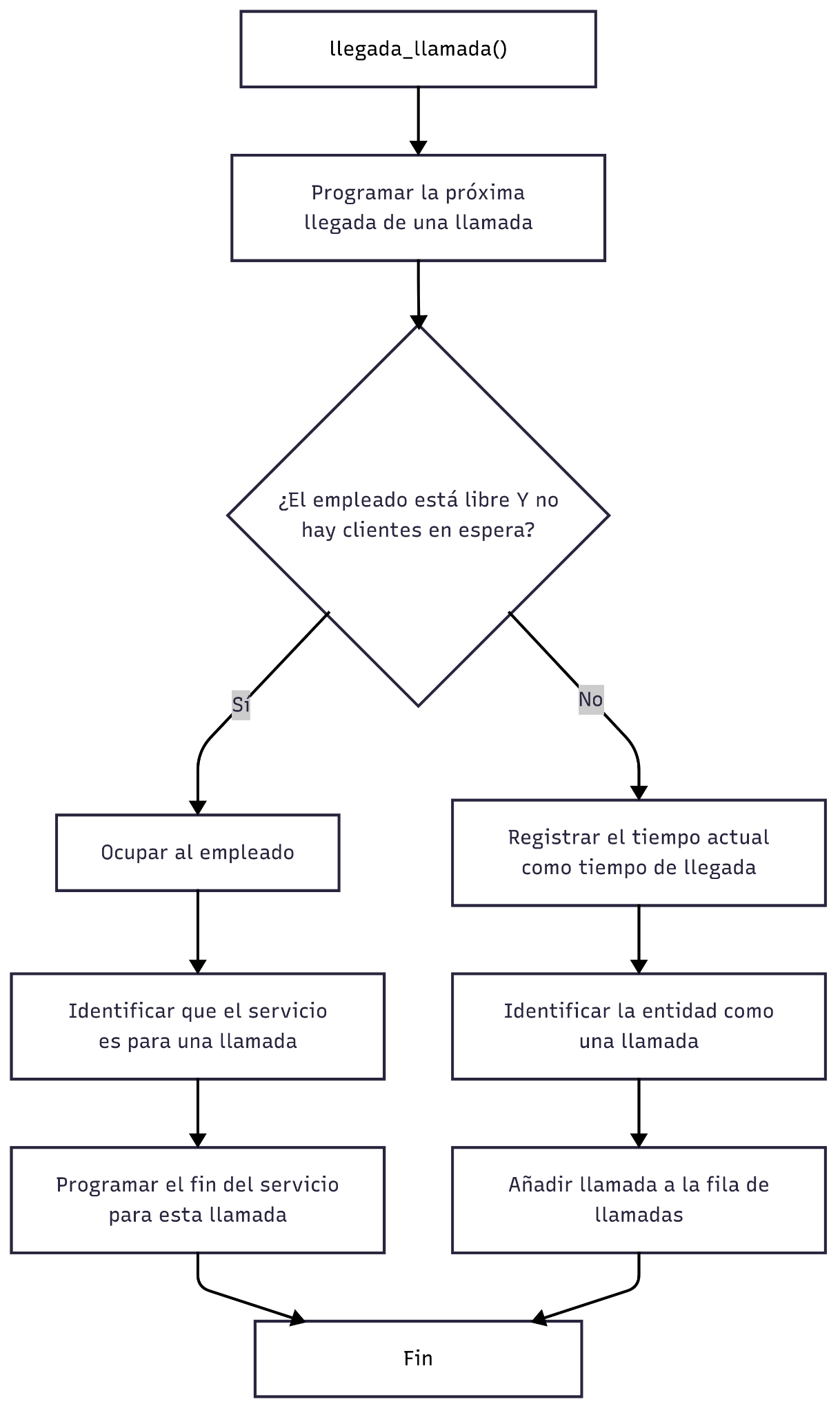
#### Rutina principal (model()):



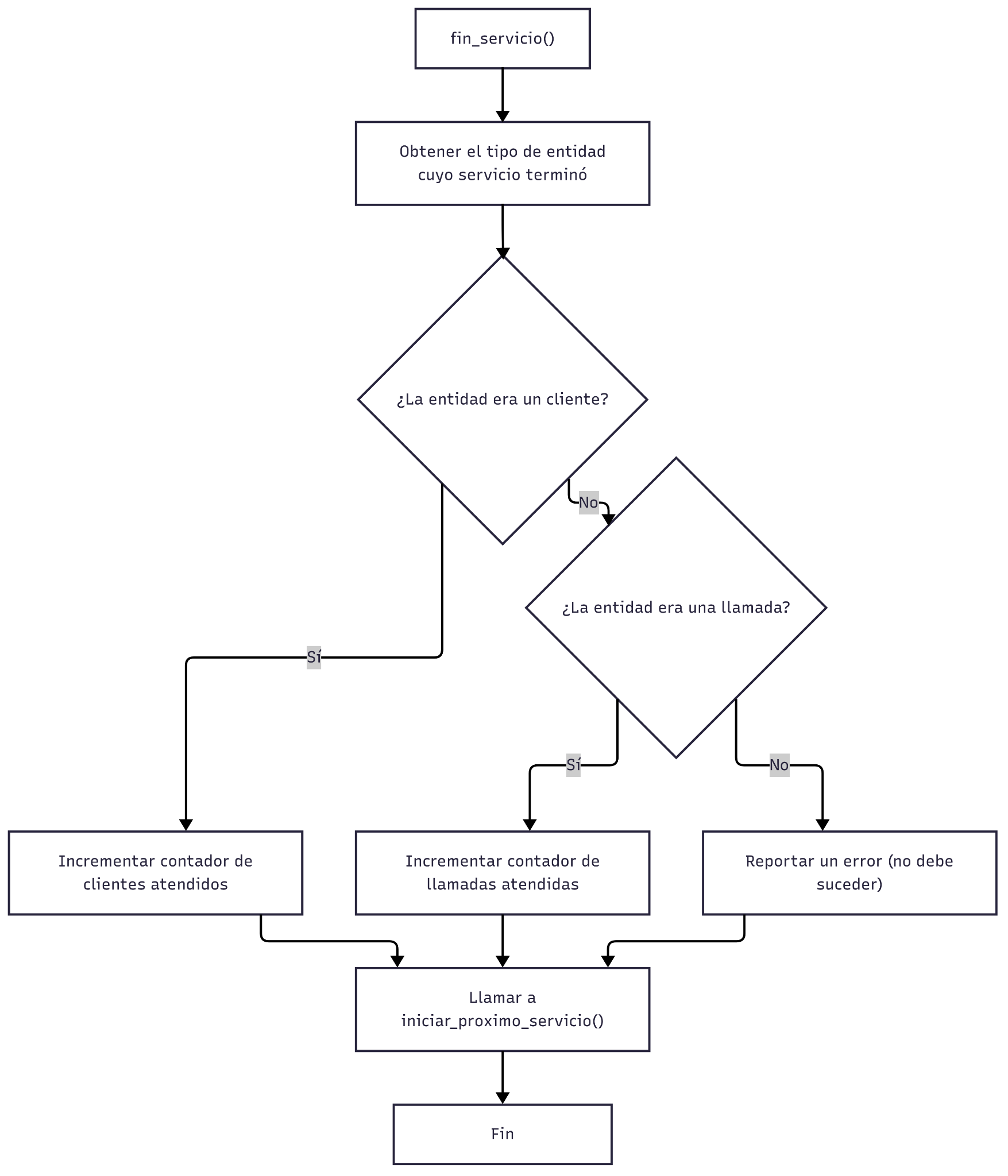
#### llegada\_cliente():



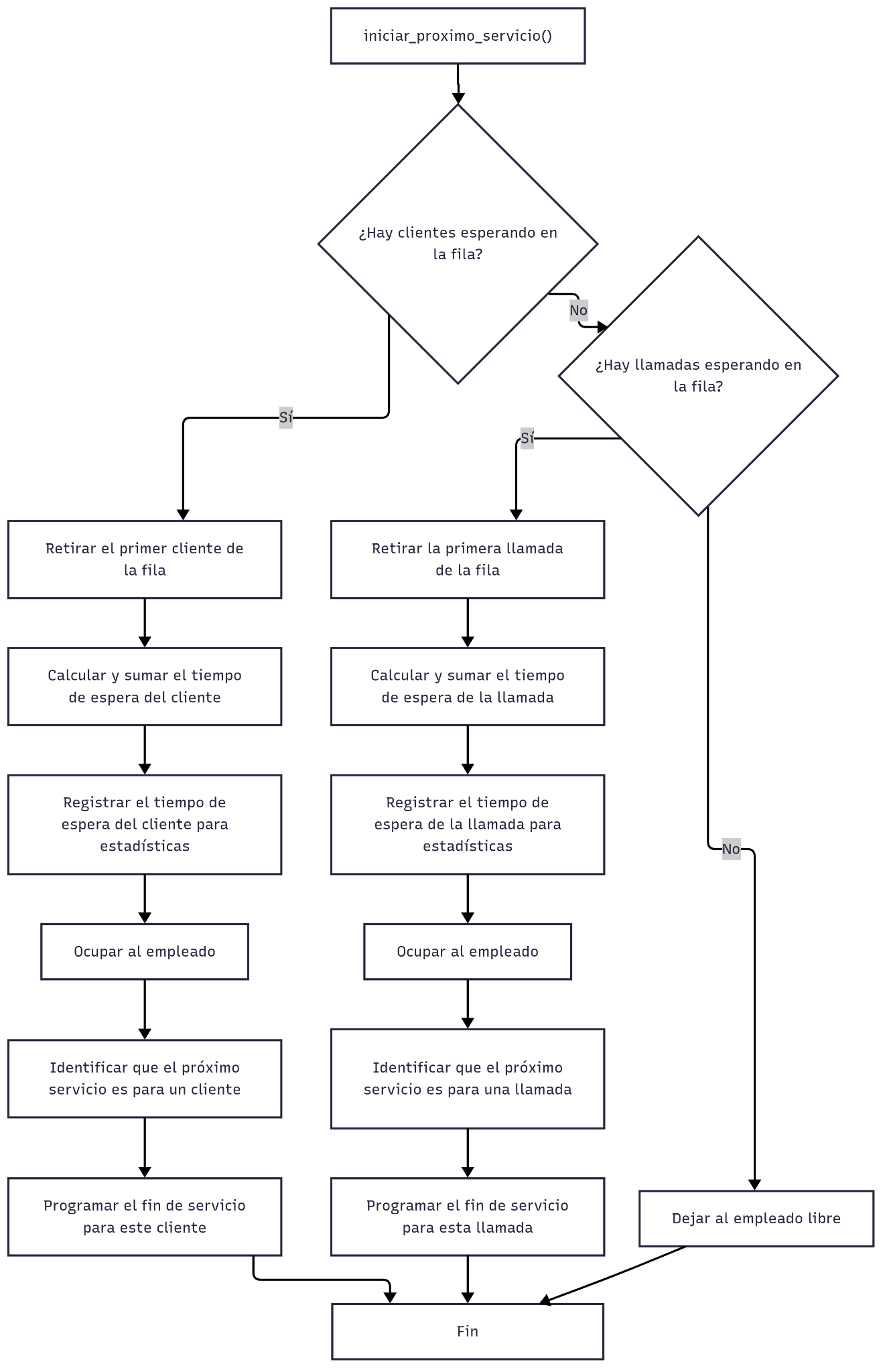
#### llegada\_llamada():



#### fin\_servicio():



#### iniciar\_proximo\_servicio():



### Análisis de resultados

Para el análisis de los resultados se empleó una semilla (de número 44444) para la generación de las variables aleatorias, indicada en el archivo simlib.c. Los resultados obtenidos son los siguientes:

RESULTADOS SIMULACION TEATRO:

Tiempo de simulacion: 8.00 horas (480.00 minutos)

Media de tiempo entre llegadas de clientes (min): 12.00

Media de tiempo de servicio de clientes (min): 6.00

Media de tiempo entre llegadas de llamadas (min): 10.00

Media de tiempo de servicio de llamadas (min): 5.00

Medidas de Desempeno

Tiempo de espera promedio de clientes en persona (min): 7.80

Tiempo de espera promedio de llamadas (min): 18.05

Numero total de clientes atendidos: 40

Numero total de llamadas atendidas: 34

Total de servicios completados: 74

Estadisticas de Tiempos de Espera (sampst):

Clientes en persona:

Estadisticas de la muestra 1:

Observaciones: 27

Promedio: 11.5482

Desv. Est.: 12.4180

Llamadas telefónicas:

Estadisticas de la muestra 2:

Observaciones: 22

Promedio: 27.8945

Desv. Est.: 19.6384

De los indicadores de desempeño se puede ver que el tiempo de espera promedio para los clientes en persona es significativamente menor al tiempo de espera promedio para las llamadas, lo cuál se explica fácilmente por el sistema de prioridad del empleado del teatro. Recordemos que las llamadas solo son atendidas cuando no hay clientes en persona en cola, por lo que sí hay clientes en la fila del teatro las llamadas se postergan hasta que el empleado termine de atender a todos los clientes en persona. Además, se puede observar que por este mismo sistema de prioridad el empleado logra darle servicio a más clientes que llamadas; sin embargo, es curioso que la distribución entre clientes y llamadas que fueron atendidas no tiene una gran diferencia, pues solamente se atendieron 6 clientes más que llamadas a pesar de que las llamadas tuvieron en promedio un tiempo de espera más que el doble del tiempo de espera de los clientes en persona. Esto se explica gracias a los tiempos de servicio de cada tipo de atención; como la media de la distribución exponencial de los tiempos de servicio de las llamadas es menor que la media de los tiempos de servicio de los clientes en persona, el empleado aunque le dé prioridad a los clientes físicamente presentes en el teatro, logra las llamadas de manera más rápida.

### Planteamiento de modificaciones

A partir de los resultados obtenidos y analizados anteriormente, la modificación que se plantea para el sistema, de modo que mejoren (se reduzcan) los tiempos de espera de los clientes y de las llamadas es que haya más de un empleado. En este caso, con un segundo empleado que se dedique exclusivamente a tomar llamadas los tiempos de espera de las personas que llaman se va a ver afectado positivamente de manera significativa. En este caso el primer empleado solamente atiende clientes en persona y no se preocupa por las llamadas cuando no hayan clientes en la fila, y el nuevo empleado no se preocupa en absoluto por los clientes en persona, puesto que él solo tomará llamadas. Tras implementar los cambios en el modelo, se obtuvieron los siguientes resultados:

REPORTE DE SIMULACION DE TEATRO (MEJORA: 2 EMPLEADOS)

Parametros de Entrada:

Tiempo de Simulacion: 8.00 horas (480.00 minutos)

Numero de Empleados: 2

Media Llegada Clientes: 12.00 min

Media Servicio Clientes: 6.00 min

Media Llegada Llamadas: 10.00 min

Media Servicio Llamadas: 5.00 min

Politica de Prioridad: Llamadas con prioridad alta (siempre que haya una en cola).

Medidas de Desempeño:

Tiempo Promedio Espera Clientes: 3.20 min

Tiempo Promedio Espera Llamadas: 1.65 min

Total Clientes Atendidos: 46

Total Llamadas Atendidas: 45

Total Servicios Completados: 91

Estadisticas Detalladas (sampst):

Tiempos de Espera Clientes:

Estadisticas de la muestra 1:

Observaciones: 16

Promedio: 9.2020

Desv. Est.: 7.8361

Tiempos de Espera Llamadas:

Estadisticas de la muestra 2:

Observaciones: 15

Promedio: 4.9374

Desv. Est.: 3.2403

Como se puede observar los tiempos de espera para ambas filas (clientes presenciales y llamadas telefónicas) mejoraron significativamente, y fueron más la cantidad de clientes atendidos para ambas categorías y en total. El código que implementa estos cambios se encuentra en el mismo Colab con la primer implementación ([ejercicio-2-4.ipynb](https://colab.research.google.com/drive/15yx7Xx77y724t21SSMJwN-32sng8NZOz?authuser=1#scrollTo=u9fQa4kw--th&line=1&uniqifier=1)) y en la carpeta compartida de Drive ([ejercicio-2-4-SIMLIB](https://drive.google.com/drive/folders/1ZHf1ZjVl6q19EaCyWPzKVhBsIWDoCITN?usp=sharing)).

## Desarrollo en SimPy

### Parámetros de entrada

Los valores que definen el comportamiento del sistema y la duración de la simulación son:

* **Tiempo de Simulación**: La duración total de la simulación, fijada en 8 horas (o 480 minutos).
* **Media de Tiempo entre Llegadas de Clientes en Persona**: Los clientes en persona llegan siguiendo una distribución exponencial con una media de 12 minutos entre llegadas.
* **Media de Tiempo de Servicio de Clientes en Persona**: El tiempo que el empleado dedica a cada cliente en persona sigue una distribución exponencial con una media de 6 minutos.
* **Media de Tiempo entre Llegadas de Llamadas Telefónicas**: Las llamadas telefónicas llegan siguiendo una distribución exponencial con una media de 10 minutos entre llamadas.
* **Media de Tiempo de Servicio de Llamadas Telefónicas**: El tiempo que el empleado dedica a cada llamada sigue una distribución exponencial con una media de 5 minutos.
* **Primera Llegada de Cliente**: El primer cliente en persona llega al sistema a los 2 minutos de iniciado el día.
* **Primera Llegada de Llamada**: La primera llamada telefónica llega al sistema a los 3 minuto**s** de iniciado el día.

### Variables del modelado

En SimPy, el estado del sistema se gestiona principalmente a través de los objetos del entorno y los recursos.

* **Entorno de Simulación (env)**: Es el corazón de la simulación, gestiona el tiempo y la programación de los eventos.
* **Empleado (empleado)**: Se modela como un PriorityResource de SimPy con una capacity=1. Este recurso representa al único empleado del teatro. SimPy se encarga internamente de gestionar si el empleado está ocupado o libre, y de mantener la cola de solicitudes para el empleado.

### Descripción del evento y tipo de evento

Los eventos clave en esta simulación son:

* **Llegada de Entidad**: Un nuevo cliente en persona o una nueva llamada telefónica llega al sistema. Esto es gestionado por los generadores de llegadas.
* **Solicitud de Recurso**: Una entidad (cliente o llamada) intenta "solicitar" el uso del empleado. Si el empleado está ocupado, la entidad entra en una cola.
* **Inicio de Servicio**: Una entidad es atendida por el empleado después de haber solicitado y obtenido el recurso.
* **Fin de Servicio**: Una entidad termina de ser atendida por el empleado, liberando el recurso.
* **Fin de Simulación**: El momento en que la simulación se detiene, definido por el until en env.run().

**Prioridad de Atención:** La prioridad se define al solicitar el recurso empleado mediante el parámetro priority:

* Los **clientes en persona** solicitan el empleado con priority=1.
* Las **llamadas telefónicas** solicitan el empleado con priority=2.

En SimPy, un número de prioridad **más bajo** indica una **mayor prioridad**. Por lo tanto, en esta simulación, los **clientes en persona tienen mayor prioridad** que las llamadas telefónicas. Si el empleado está libre y hay clientes y llamadas esperando, un cliente será atendido primero.

### Listas y sus atributos

En SimPy, las colas son gestionadas automáticamente por los objetos Resource (en este caso, empleado). No se requiere una implementación manual de listas enlazadas como en SIMLIB/C.

* El empleado tiene una **cola interna** donde las entidades esperan si el recurso no está disponible.
* Los "atributos" de las entidades (como el tiempo de llegada) son variables locales dentro de cada proceso (cliente o llamada) y se utilizan para cálculos específicos de esa entidad.

### Contadores y/o acumuladores

Para recopilar datos y calcular las medidas de desempeño, la simulación utiliza listas de Python para almacenar los tiempos de espera individuales:

* **tiempos\_espera\_clientes**: Una lista que almacena el tiempo que cada cliente en persona esperó desde su llegada hasta que el empleado comenzó a atenderlo.
* **tiempos\_espera\_llamadas**: Una lista que almacena el tiempo que cada llamada telefónica esperó desde su llegada hasta que el empleado comenzó a atenderla.
* El **número total de clientes atendidos** y **llamadas atendidas** se obtiene directamente contando la cantidad de elementos en estas listas al final de la simulación (len(tiempos\_espera\_clientes) y len(tiempos\_espera\_llamadas)).

### Medidas de desempeño

Al finalizar la simulación, el código proporcionado calcula y presenta las siguientes medidas clave para evaluar el rendimiento del sistema:

* **Tiempo de Espera Promedio de Clientes**: Se calcula como la media de todos los tiempos de espera registrados en la lista tiempos\_espera\_clientes (np.mean(tiempos\_espera\_clientes)).
* **Tiempo de Espera Promedio de Llamadas**: Se calcula como la media de todos los tiempos de espera registrados en la lista tiempos\_espera\_llamadas (np.mean(tiempos\_espera\_llamadas)).
* **Total Clientes Atendidos (con espera)**: Indica el número total de clientes en persona que fueron atendidos y para los cuales se registró un tiempo de espera. Se obtiene a partir de la longitud de la lista tiempos\_espera\_clientes.
* **Total Llamadas Atendidas (con espera)**: Indica el número total de llamadas telefónicas que fueron atendidas y para las cuales se registró un tiempo de espera. Se obtiene a partir de la longitud de la lista tiempos\_espera\_llamadas.
* **Total Servicios Completados (con espera)**: Representa la suma total de clientes en persona y llamadas telefónicas que completaron su servicio (es decir, fueron atendidos tras potencialmente haber esperado) durante la duración de la simulación. Esta medida se obtiene al sumar las longitudes de las listas tiempos\_espera\_clientes y tiempos\_espera\_llamadas.

### Subprogramas y propósito

El modelo está organizado en funciones generadoras (procesos) y funciones auxiliares que definen el comportamiento de las entidades y la lógica de la simulación:

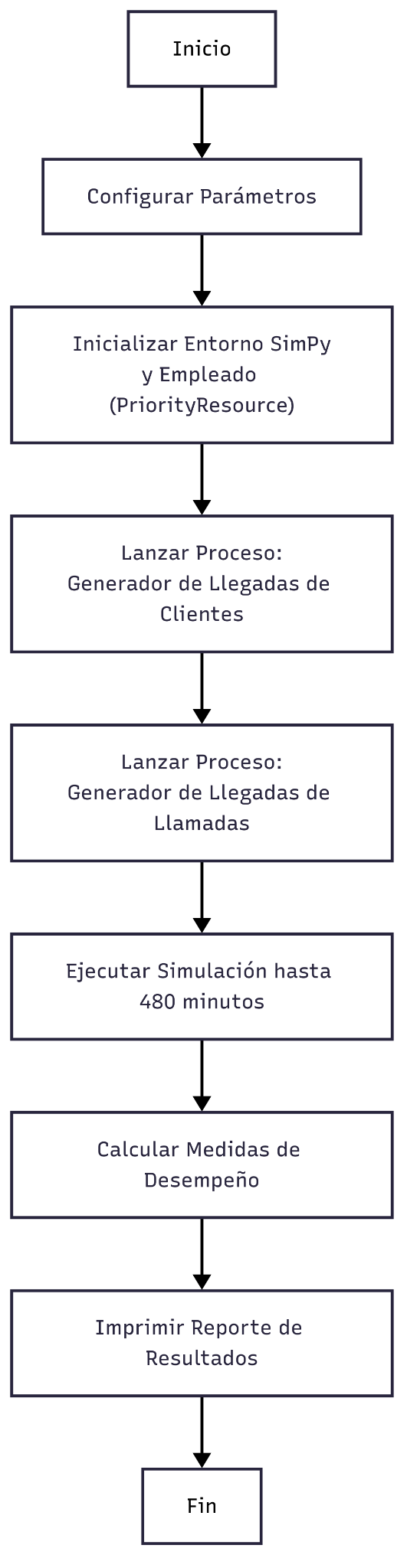
* **cliente(env, empleado)**: Define el comportamiento de un cliente individual. Este proceso llega, solicita el empleado, espera si es necesario, es atendido y luego finaliza.
* **llamada(env, empleado)**: Define el comportamiento de una llamada telefónica individual. Similar al cliente, pero con sus propios tiempos de llegada/servicio y prioridad.
* **generador\_llegadas\_clientes(env, empleado)**: Un proceso que se encarga de programar la llegada de nuevos clientes en persona a lo largo de la simulación, según la distribución de tiempo entre llegadas.
* **generador\_llegadas\_llamadas(env, empleado)**: Un proceso similar al anterior, pero para la llegada de nuevas llamadas telefónicas.
* **Bloque Principal de Ejecución**: La parte final del script que configura el entorno de SimPy (simpy.Environment()), crea el recurso empleado (simpy.PriorityResource()), inicia los procesos generadores de llegadas (env.process()) y ejecuta la simulación hasta el tiempo definido (env.run()), para finalmente imprimir el reporte de resultados.

### Implementación con SimPy

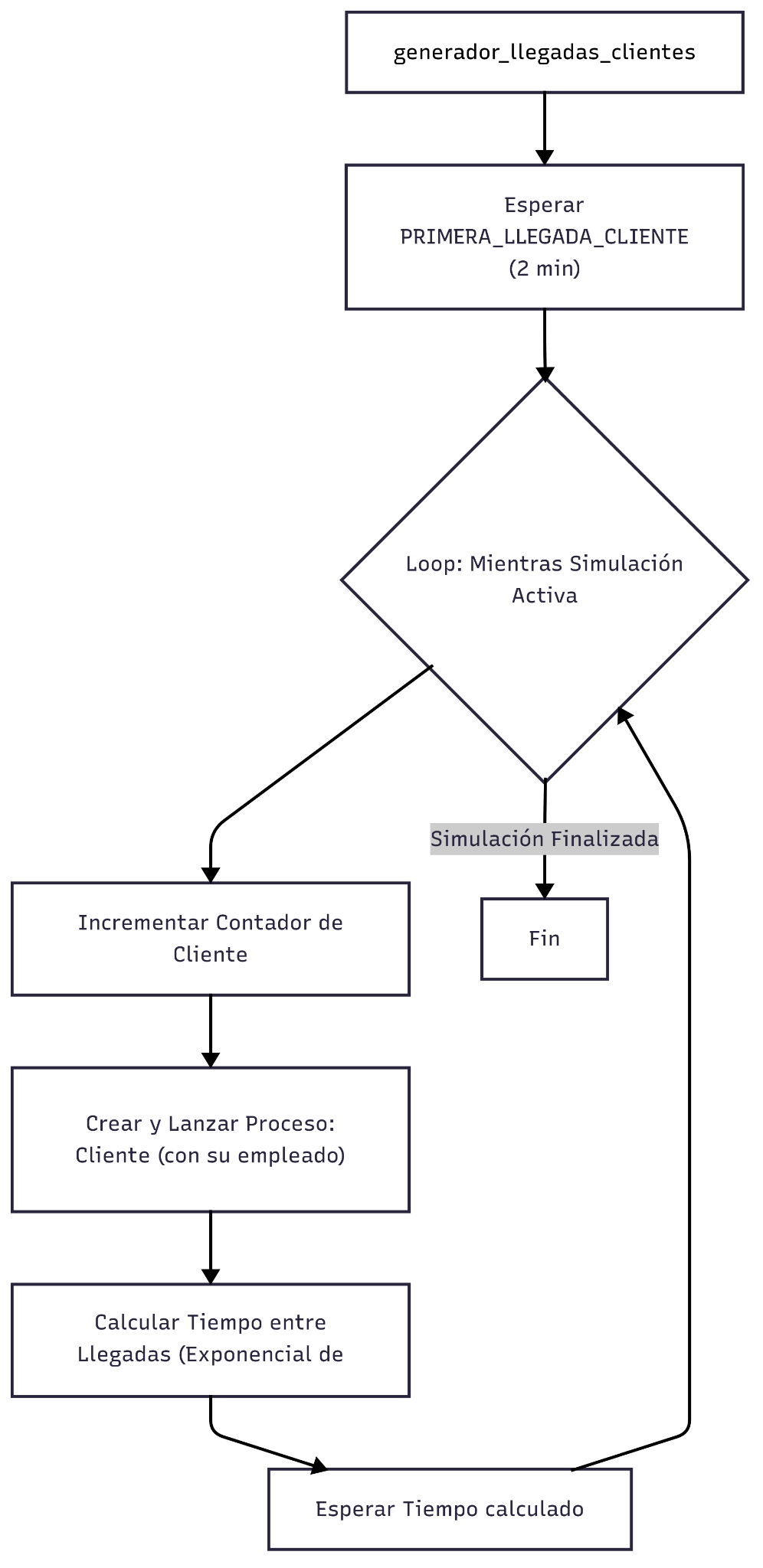
La implementación de la simulación en Python, utilizando SimPy [2], se encuentra en el siguiente Colab: [ejercicio-2-4.ipynb](https://colab.research.google.com/drive/15yx7Xx77y724t21SSMJwN-32sng8NZOz?authuser=1#scrollTo=63g7aqs1ZBFZ&line=1&uniqifier=1)

### Diagramas de flujo

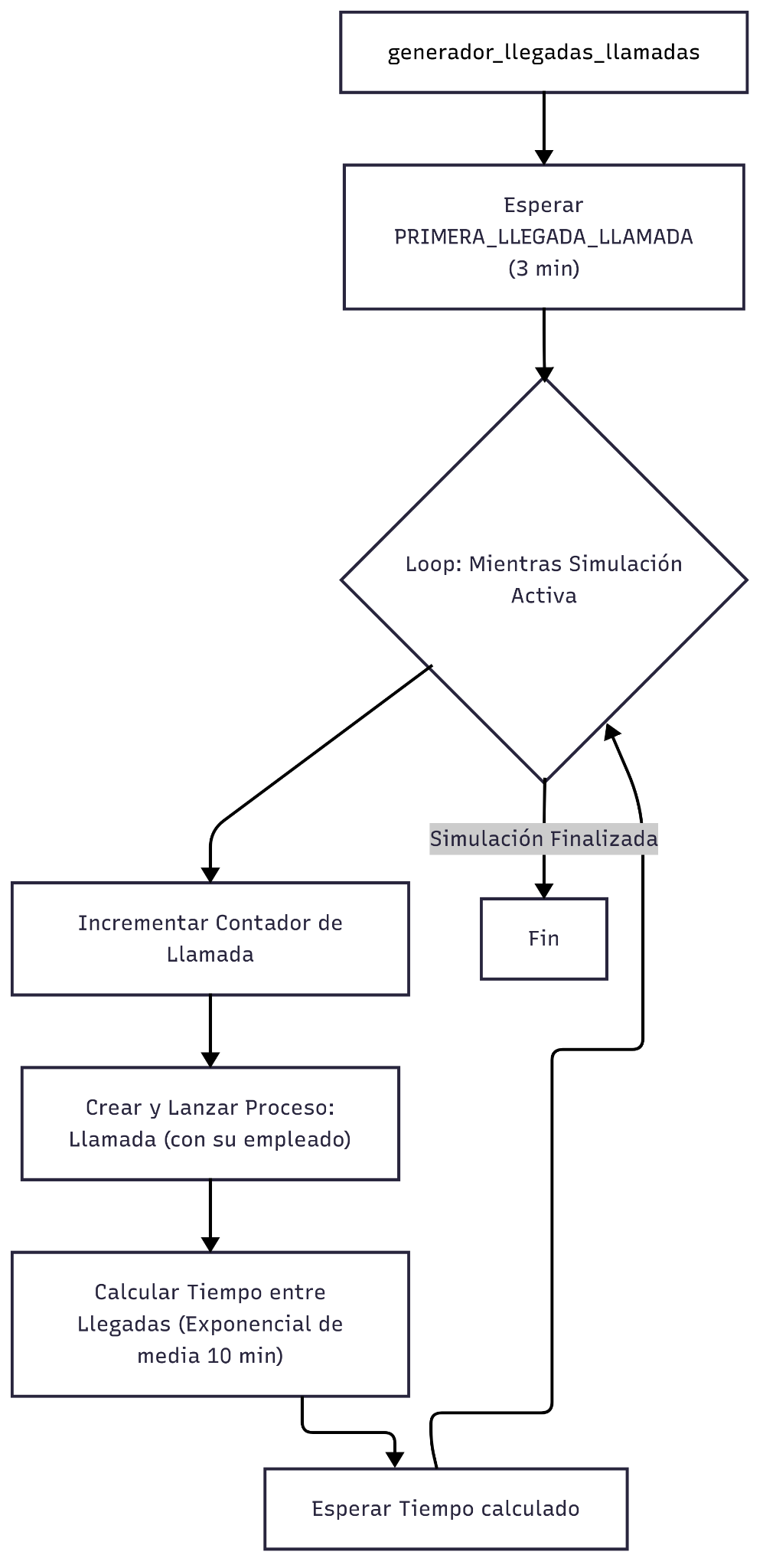
Rutina principal:



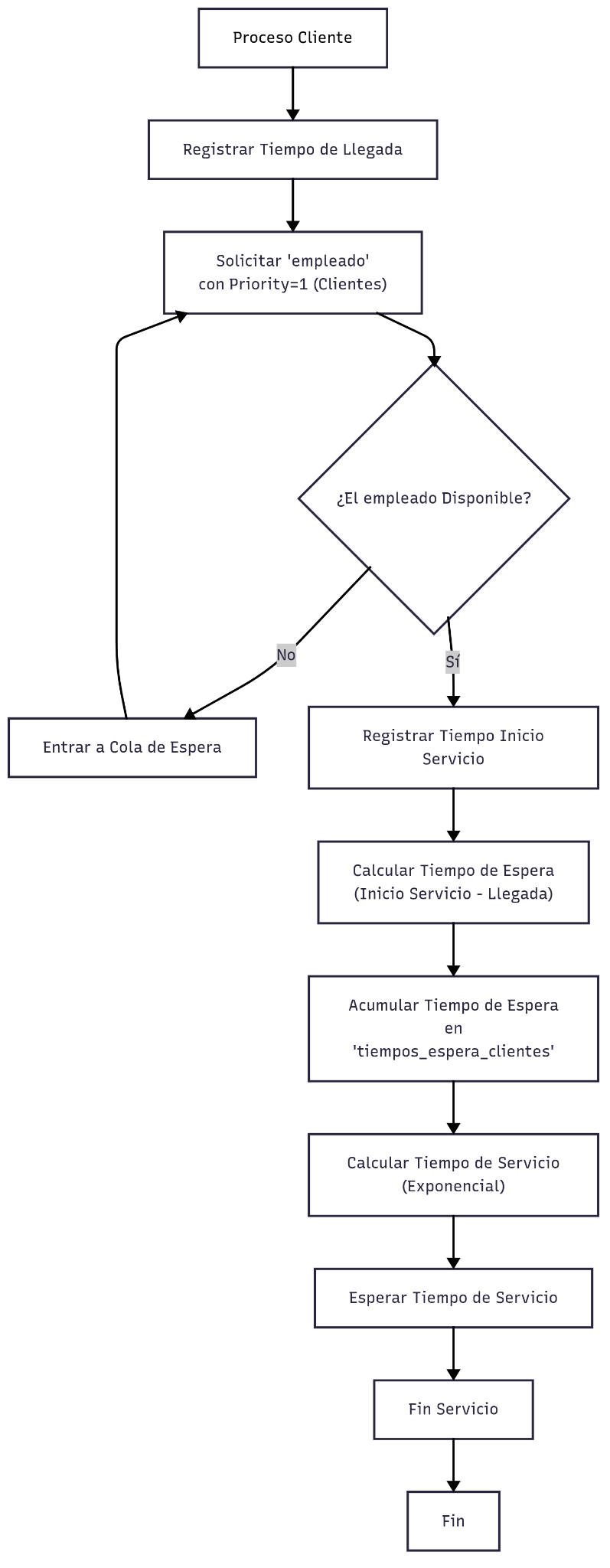
generador\_llegadas\_clientes:



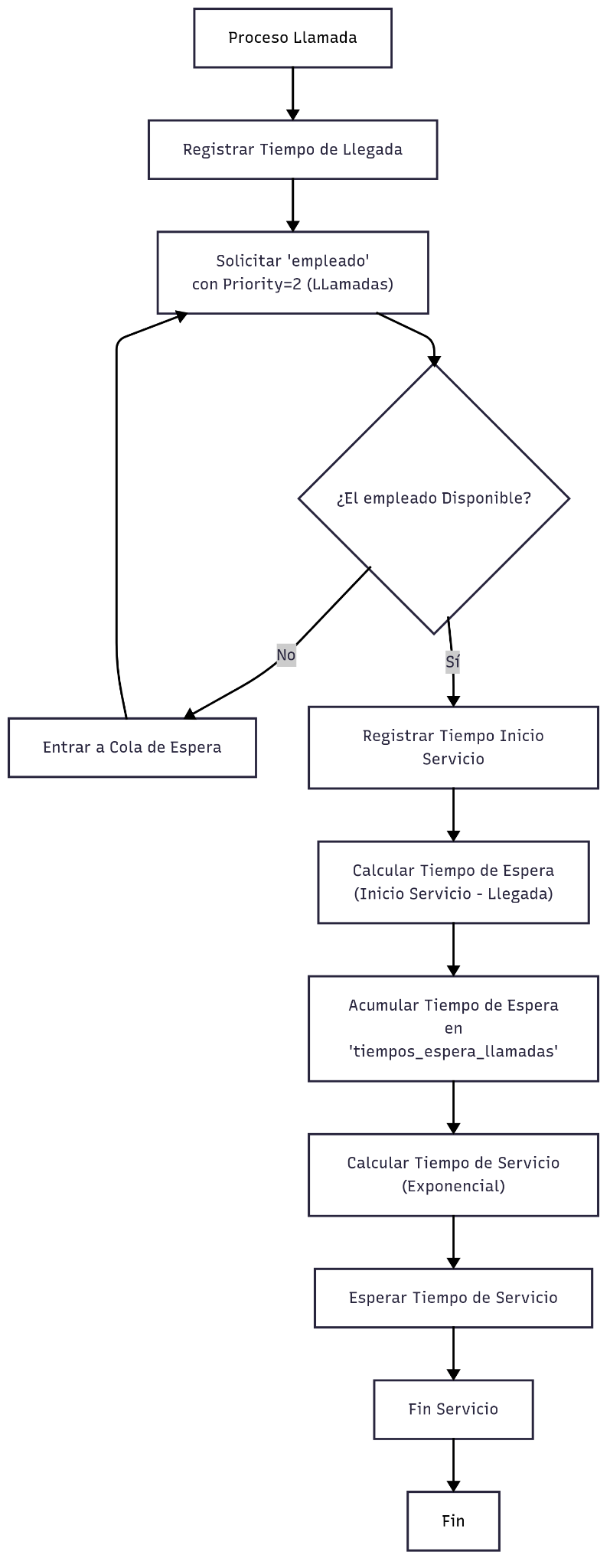
generador\_llegadas\_llamadas:



cliente:



llamada:



### Análisis de resultados

Para el análisis de los resultados se empleó una semilla (de número 44) para la generación de las variables aleatorias, haciendo random.seed(80) antes de hacer el run de la simulación. Los resultados obtenidos son los siguientes:

RESULTADOS SIMULACION TEATRO

Tiempo de Simulación: 8.0 horas (480.0 minutos)

Llegada Clientes (Media): 12.0 min

Servicio Clientes (Media): 6.0 min

Llegada Llamadas (Media): 10.0 min

Servicio Llamadas (Media): 5.0 min

Tiempo de espera promedio de Clientes: 13.00 minutos

Tiempo de espera promedio de Llamadas: 37.99 minutos

Total Clientes atendidos (con espera): 39

Total Llamadas atendidas (con espera): 31

Total servicios completados (con espera): 70

De los indicadores de desempeño se puede ver, una vez más, que el tiempo de espera promedio para los clientes en persona es significativamente menor al tiempo de espera promedio para las llamadas, lo cuál se explica fácilmente por el sistema de prioridad del empleado del teatro. También se puede observar, nuevamente, que por este mismo sistema de prioridad el empleado logra darle servicio a más clientes que llamadas; sin embargo, es curioso que la distribución entre clientes y llamadas que fueron atendidas no tiene una gran diferencia, pues solamente se atendieron 8 clientes más que llamadas a pesar de que las llamadas tuvieron en promedio un tiempo de espera casi igual a tres veces el tiempo de espera de los clientes en persona. Esto se explica gracias a los tiempos de servicio de cada tipo de atención; como la media de la distribución exponencial de los tiempos de servicio de las llamadas es menor que la media de los tiempos de servicio de los clientes en persona, el empleado aunque le dé prioridad a los clientes físicamente presentes en el teatro, logra darle servicio a las llamadas de manera más rápida.

### Planteamiento de modificaciones

A partir de los resultados obtenidos y analizados anteriormente, la modificación que se plantea para el sistema, de modo que mejoren (se reduzcan) los tiempos de espera de los clientes y de las llamadas es, nuevamente, que haya más de un empleado. En este caso, con un segundo empleado que se dedique exclusivamente a tomar llamadas los tiempos de espera de las personas que llaman se va a ver afectado positivamente de manera significativa. De esta manera, el primer empleado solamente atiende clientes en persona y no se preocupa por las llamadas cuando no haya clientes en la fila, y el nuevo empleado no se preocupa en absoluto por los clientes en persona, puesto que él solo tomará llamadas. Tras implementar los cambios en el modelo, se obtuvieron los siguientes resultados:

RESULTADOS SIMULACION TEATRO

Tiempo de Simulación: 8.0 horas (480.0 minutos)

Llegada Clientes (Media): 12.0 min

Servicio Clientes (Media): 6.0 min

Llegada Llamadas (Media): 10.0 min

Servicio Llamadas (Media): 5.0 min

Configuración: Dos empleados (uno para cada cola)

Tiempo de espera promedio de Clientes: 2.50 minutos

Tiempo de espera promedio de Llamadas: 14.09 minutos

Total Clientes atendidos (con espera): 35

Total Llamadas atendidas (con espera): 52

Total servicios completados (con espera): 87

Como se puede observar los tiempos de espera para ambas filas (clientes presenciales y llamadas telefónicas) mejoraron significativamente, una vez más, gracias a este cambio en el modelo. En esta ocasión no se atendieron más clientes pero sí muchas más llamadas (más de 20 llamadas adicionales), lo cuál se explica porque con una fila exclusiva para atender las llamadas, que toman menos tiempo y llegan más seguido por lo general, es mayor la productividad del nuevo empleado. El código que implementa estos cambios se encuentra en el mismo Colab con la primer implementación ([ejercicio 2.4.ipynb](https://colab.research.google.com/drive/15yx7Xx77y724t21SSMJwN-32sng8NZOz?authuser=1#scrollTo=-hxfdBTY3AA8)).

# Ejercicio 2.5

Una pequeña bodega se surte de material usando una flotilla de 3 camionetas. Se contrata a un muchacho para descargar las camionetas. El tiempo de descarga depende del número de camionetas en la bodega(n) de acuerdo con la función E (t) = 1 /n horas para n= 1, 2,3 con distribución exponencial. Las camionetas descargadas van por más material y el tiempo de viaje es de 2 horas con distribución exponencial.

Determine:

1. El número promedio de camionetas esperando ser descargadas.
2. La probabilidad de que el muchacho esté ocupado
3. La probabilidad de que todas las camionetas estén de viaje.
4. El número promedio de camionetas viajando.

## 

### Parámetros de entrada

| Parámetro | Distribución | Valor |
| --- | --- |
| Número de camionetas | 3 unidades |
| Tiempo de descarga | Exponencial, |
| Tiempo de viaje | Exponencial, con media 2 horas |
| Horizonte de simulación | Definido por el usuario |

### Variables del modelado

| Variable | Descripción |
| --- | --- |
| t | Reloj de simulación |
| n\_waiting | Camionetas esperando descarga |
| N\_in\_service (Operario) | Camionetas siendo descargadas (0 ó 1) |
| n\_traveling | Camionetas en viaje |

### Descripción del evento y tipo de evento

| Evento | Descripción |
| --- | --- |
| ArriboVeh | Una camioneta llega de viaje y entra a la cola de descarga (n\_waiting++). |
| FinDescarga ( y Programación de viaje) | Muchacho termina de descargar una camioneta: muchacho se libera y programa viaje. |
| FinViaje | Una camioneta finaliza su viaje y genera un ArriboVeh inmediato. |

### Listas y sus atributos

| Lista | Contenido |
| --- | --- |
| FEL (Future Event List) | Estructura ordenada por tiempo de evento (Tiempo , evento) |
| Cola descarga | FIFO de camionetas esperando; cada elemento registra veh\_id. |

### Contadores y/o acumuladores

| Acumulador | Descripción |
| --- | --- |
| area\_waiting |  |
| area\_traveling |  |
| area\_all\_traveling |  |
| t\_busy\_operario | Tiempo que el operario está ocupado |
| count\_descargas | Número total de descargas completadas |
| sum\_waiting\_time | Suma de tiempos de espera en cola de descarga |

### Medidas de desempeño

| Número promedio de camionetas esperando descarga |  |
| --- | --- |
| Número promedio de camionetas en viaje |  |
| Tiempo promedio de espera en cola |  |
| Probabilidad de que todas las camionetas estén viajando |  |
| Utilización del operario |  |

### Subprogramas y propósito

* **Init()**: Inicializa variables, calendario y programa evento FinViaje para camionetas iniciales.
* **Schedule(event)**: Inserta un nuevo evento en la FEL ordenada por tiempo.
* **ArriboVeh()**: Maneja las llegadas de las camioneta, incrementa n\_waiting. Sí está el operario libre inicia la descarga.
* **FinDescarga()**: Al finalizar la descarga, libera operario, decrementa n\_waiting, programa FinViaje y se actualiza el acumulador.
* **FinViaje()**: Al finalizar viaje se incrementa n\_waiting y decrementa n\_traveling.
* **Run()**: Inicia la simulación llamando a Init(), ejecuta el ciclo de eventos hasta el tiempo determinado T, y llama a Report().
* **Report()**: Calcula y muestra métricas de desempeño al terminar la simulación.

### 

## Desarrollo en SimPy y Desarrollo en SIMLIB

Simpy

[Ejercicio 2.5.ipynb](https://colab.research.google.com/drive/1eBRR1Byznmc79MwnTnnAwW7eu7JBxVcn#scrollTo=No5Va35ba2Et)

Simlib

[Ejercicio 2.5.ipynb](https://colab.research.google.com/drive/1eBRR1Byznmc79MwnTnnAwW7eu7JBxVcn#scrollTo=F4ss4qUgA8ag)

## Análisis de resultados

## Los resultados obtenidos en el caso de la simulación en Simpy son: (Muy similares a los de Simlib)

El sistema opera con un nivel de eficiencia notable, apenas 0.39 camionetas en promedio esperan en la cola de descarga, esto demuestra que el operario atiende ágilmente y evita cuellos de botella. Además, 1.92 camionetas circulan en viaje, aprovechando bastante bien la flota disponible. El tiempo medio de espera en cola es de 0.45 horas (aproximadamente 27 min), un valor que refleja un flujo de trabajo fluido, mientras que el operario está ocupado el 68.9 % del tiempo, confirmando una alta utilización sin sobrecarga excesiva. Además, en el 31.1 % del tiempo de simulación las tres camionetas se encuentran simultáneamente en viaje, lo cual tiene sentido si evaluamos la utilización del operario

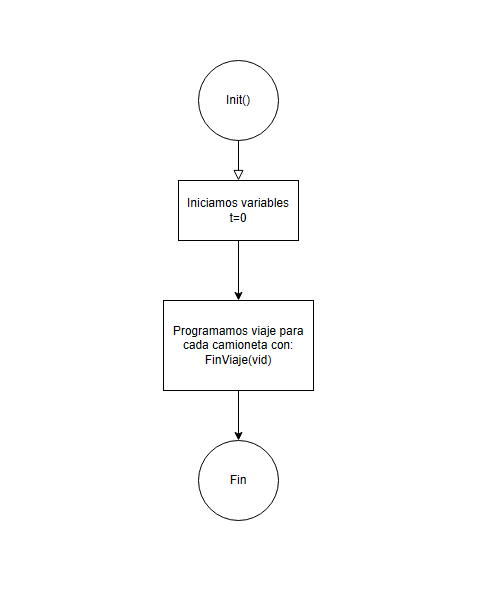
**Planteamiento de modificaciones.**

Podríamos evaluar el impacto de sumar un segundo operario para atender descargas en paralelo, lo cual reduciría directamente la espera en cola y permitiría descargar más camionetas de retorno, de igual forma, incrementar la flota a 4 o 5 unidades disminuiría la probabilidad de falta de vehículos disponibles (mejorando la cobertura) y repartiría mejor el tiempo de viaje entre más activos, lo que aligera la carga de cada camión y evita picos de congestión en la cola de descarga, además con esto no desaprovechamos los dos operarios disponibles. Con esas dos mejoras (más manos para descargar y más camionetas para distribuir el trabajo) se puede optimizar la utilización de recursos, acortar tiempos de espera y aumentar la capacidad global del sistema.

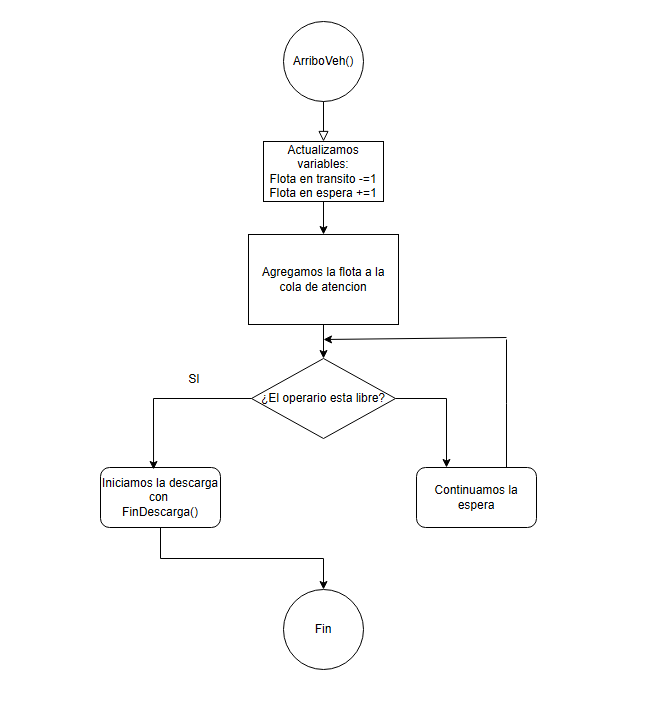
**Diagramas de flujo**

**Rutinas principales**

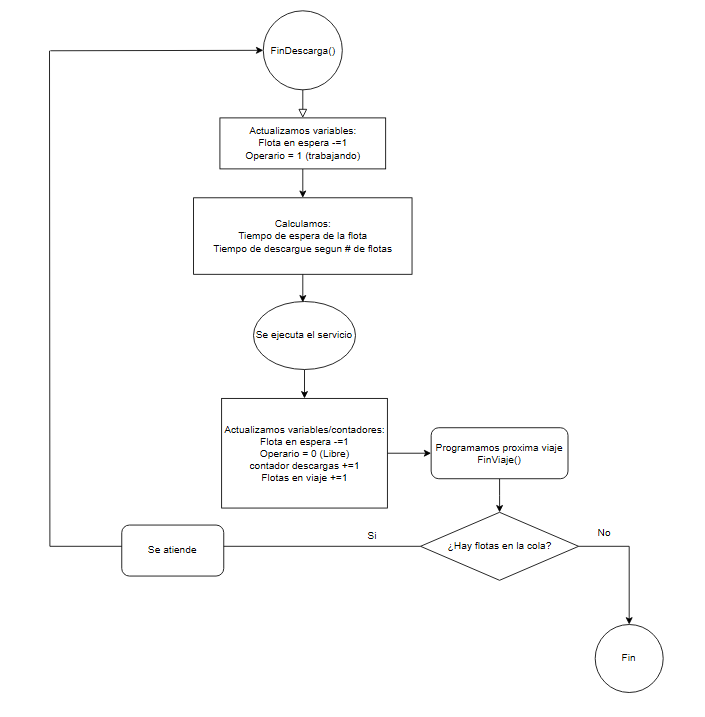
* **init()**

****

* **ArriboVeh()**

****

* **FinDescarga()**

****

# Ejercicio 2.6

Una cadena hotelera tiene dos buses para recoger y dejar personas en un aeropuerto local y dos hoteles separados. Los buses viajan desde el aeropuerto al hotel 1, luego al hotel 2, y regresan al aeropuerto para continuar con este patrón. El tiempo de viaje entre cada lugar sigue una distribución normal con una media de 20 y una desviación estándar de 2 minutos. El tiempo de llegada de los viajeros desde sus vuelos se distribuye exponencialmente con una media de 2.5 minutos. Cincuenta por ciento de las personas se bajan en el primer hotel, y el bus recoge personas de este hotel que desean ir al aeropuerto. El otro cincuenta por ciento de las personas se baja en el segundo hotel, y el bus recoge nuevamente personas. En el aeropuerto, todo el mundo se baja. En ambos hoteles las personas llegan al paradero del bus para ir al aeropuerto con tiempos entre llegadas exponenciales con media de 5 minutos. Simular el sistema donde el primer bus sale del aeropuerto al iniciar la simulación y el segundo sale del aeropuerto 30 minutos después del primero. Determine la cantidad de asientos requeridos en ambos buses tal que cualquier persona esperando pueda ser recogida.

# Ejercicio usando SimPy.

## Parámetros de entrada.

1. Tiempo de viaje de buses: el tiempo de viaje de los buses se distribuye normalmente con una *media* de 20 minutos y *desviación estándar* de 2 minutos, Los tiempos de viaje se cumplen para los trayectos: Aeropuerto → Hotel 1 → Hotel 2 → Aeropuerto.
2. Tiempo de llegada de los viajeros desde sus vuelos: El tiempo de viaje de los viajeros se distribuye exponencialmente con una *media* de 2,5 minutos. 50% de estos pasajeros se quedan en el Hotel 1 y el otro 50% se quedan en el Hotel 2
3. Tiempo llegada de personas al paradero de bus: En ambos hoteles cuando se bajan personas que llegan se suben otras que van hacia el aeropuerto, con un tiempo de llegada que se distribuye exponencialmente con una *media* de 5 minutos.
4. Tiempo de operación de los buses: El primer bus sale del aeropuerto al iniciar la simulación y el segundo bus sale 30 minutos después del primero.

## Variables del modelado.

* Capacidad del bus: Variable a determinar.
* Reloj de simulación (t = minutos transcurridos)
* Posición de los pasajeros.
* Posición del autobús:

## Descripción del evento y tipo de evento.

* Llegada pasajero aeropuerto: Un nuevo pasajero aterriza y se añade a la cola se programa según su distribución exponencial.
* Llegada pasajero parada bus Hotel 1: Un viajero quiere ir al aeropuerto y se une a la cola .
* Llegada pasajero parada bus Hotel 2: Un viajero quiere ir al aeropuerto y se une a la cola
* Llegada al Hotel 1: Descarga los pasajeros que van al Hotel 1, y carga a los pasajeros de la cola
* Llegada al Hotel 2: Descarga los pasajeros que van al Hotel 2, y carga a los pasajeros de la cola
* Llegada al aeropuerto: El completa su viaje al llegar del Hotel 2 al aeropuerto, descarga a los pasajeros que querían ir al aeropuerto, y carga a los pasajeros que están en cola
* Fin de la simulación *(evento de control)* : Se alcanza un tiempo

## Listas y sus atributos

### Pasajeros:

* + Lugar de Origen (Aeropuerto, Hotel 1, Hotel 2)
  + Tiempo de llegada a las colas ,,
  + Lugar de destino (Aeropuerto, Hotel 1, Hotel 2)

### Bus:

* + Identificador de Bus (1 o 2)
  + Capacidad ( a definir)
  + Pasajeros a bordo.
  + Tiempo de viaje entre lugares.
  + Tiempo de inicio.

### Lugares:

* + Identificador de lugares (0,1,2)
  + Cola de pasajeros ,,

## Contadores y/o acumuladores:

### Colas de espera:

* + airport\_to\_hotel1, airport\_to\_hotel2: Pasajeros esperando en el aeropuerto
  + hotel1\_to\_airport,hotel2\_to\_airport: Pasajeros esperando en los hoteles.

### Estados del bus:

* + onboard\_H1, onboard\_H2, onboard\_A: Pasajeros a bordo por destino.
  + occupancy: Número total de pasajeros en el bus.
  + space: Número de espacios libres restantes.
  + boarded\_H1, boarded\_H2: pasajeros abordados para pasarlos a onboard

### Funciones de carga y descarga:

* + update\_load: Actualiza contadores al abordar.
  + unload\_passengers: Descarga pasajeros y actualiza ocupación.

### Estados de simulación:

* + failures: Registro de eventos donde quedaron pasajeros sin transportar.
  + sim\_success: Conteo de simulaciones sin fallos.
  + rep: Número de repeticiones por configuración de capacidad.
  + capacity: Capacidad actual del bus en análisis.

## Medidas de desempeño:

### Success rate:

### **Failures**:

* + Cantidad de veces que algún bus no pudo recoger a todos los pasajeros en alguna estación. Se registra en una lista cada vez que hay más pasajeros en cola que espacios en el bus disponibles.

### Capacidad minima:

* + La capacidad mínima en los buses para que no ocurran fallos que se obtiene cuando el success rate es igual a la cantidad de repeticiones hechas en la simulación, osea cuando una capacidad al simularla no tuvo fallos.

## 

## Subprogramas y propósito:

### arrival\_airport\_p():

* **Propósito**: Genera pasajeros que llegan al aeropuerto para ser transportados a los hoteles.
* **Funcionamiento**: Alterna entre las colas de Hotel 1 y Hotel 2, generando llegadas exponenciales con media de 2.5 minutos.

### arrival\_hotel\_p():

* **Propósito**: Simula la llegada de pasajeros a un hotel que desean regresar al aeropuerto.
* **Funcionamiento**: Agrega pasajeros a la cola del hotel correspondiente usando un proceso exponencial (media = 5 minutos).

### board\_passengers():

* **Propósito**: Simula el abordaje de pasajeros desde una cola hasta que se alcance un límite.
* **Retorna**: Número de pasajeros que abordaron.

### update\_load()

* **Propósito**: Actualiza el estado del bus al subir pasajeros.
* **Retorna**: Nuevos valores de:
  + onboard: pasajeros a bordo con destino particular
  + occupancy: total de personas en el bus
  + space: asientos restantes

### unload\_passengers()

* **Propósito**: Simula la bajada de todos los pasajeros con un mismo destino.
* **Retorna**:
  + onboard: se reinicia en 0.
  + occupancy: reducido por la cantidad que bajó.

### bus\_trips()

* **Propósito**: Modelo principal que simula el ciclo completo de viaje de un bus:

1. Recolecta pasajeros en el aeropuerto
2. Visita Hotel 1 → Hotel 2
3. Regresa al aeropuerto con los pasajeros de los hoteles

* **Registra fallos** si hay más pasajeros esperando que espacio disponible.

### simulation()

* **Propósito**: Ejecuta una instancia de simulación con una capacidad específica para los buses. Este es realmente el programa inicial.
* **Inicia** todos los procesos de llegadas y viajes.
* **Retorna**: Lista de fallos registrados.

### minimum\_capacity()

* **Propósito**: Encuentra la capacidad mínima del bus que garantice que no haya fallos en todas las repeticiones.
* **Repite** la simulación con diferentes capacidades hasta encontrar la óptima.

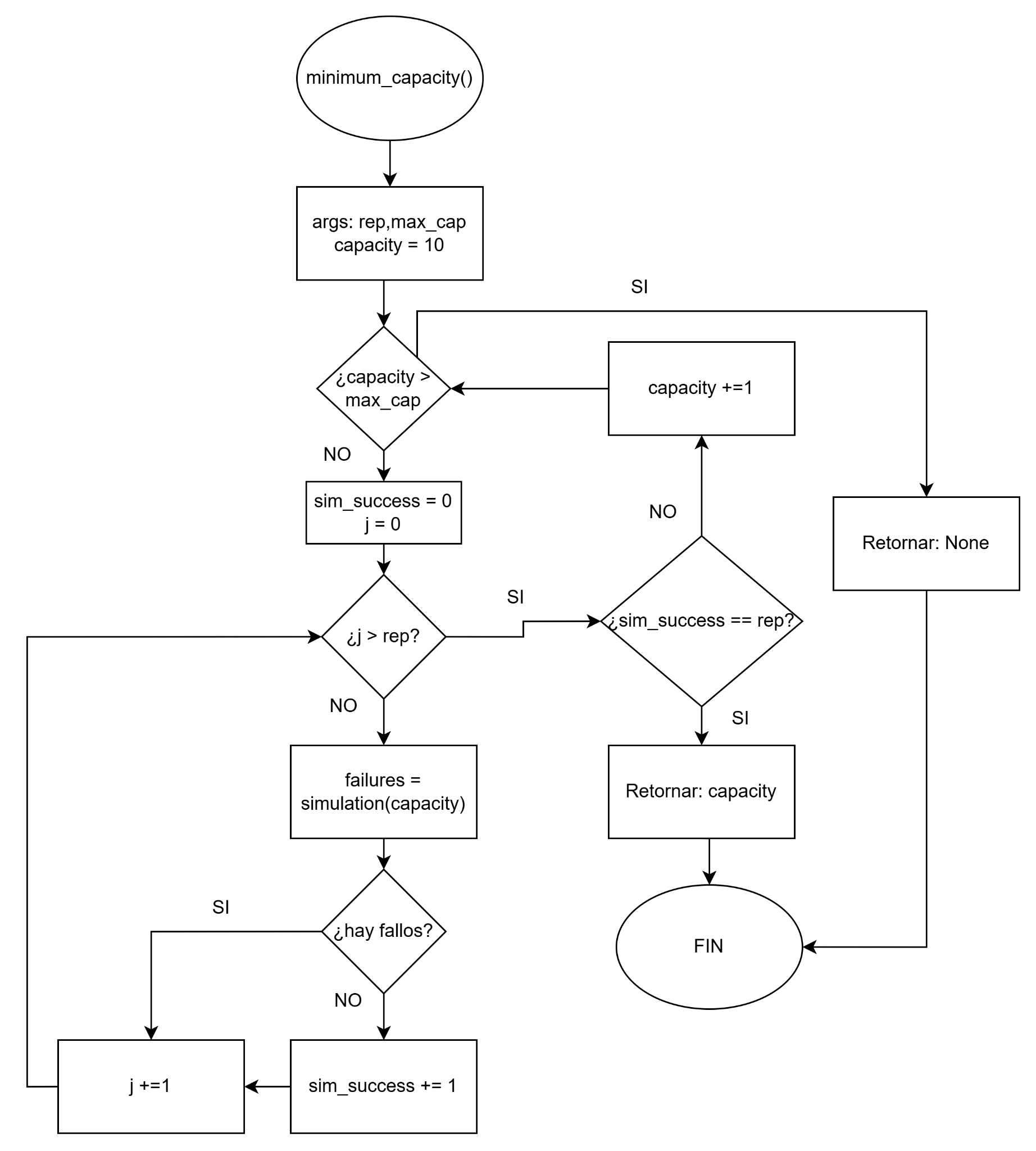
## Desarrollo en SimPy.

### Enlace al código: [Ejercicio2\_6.ipynb](https://colab.research.google.com/drive/1wB6KXpqjLmDQMqBYORGv0T2xq1pCSmLS#scrollTo=Sf4WkjIDi8yM)

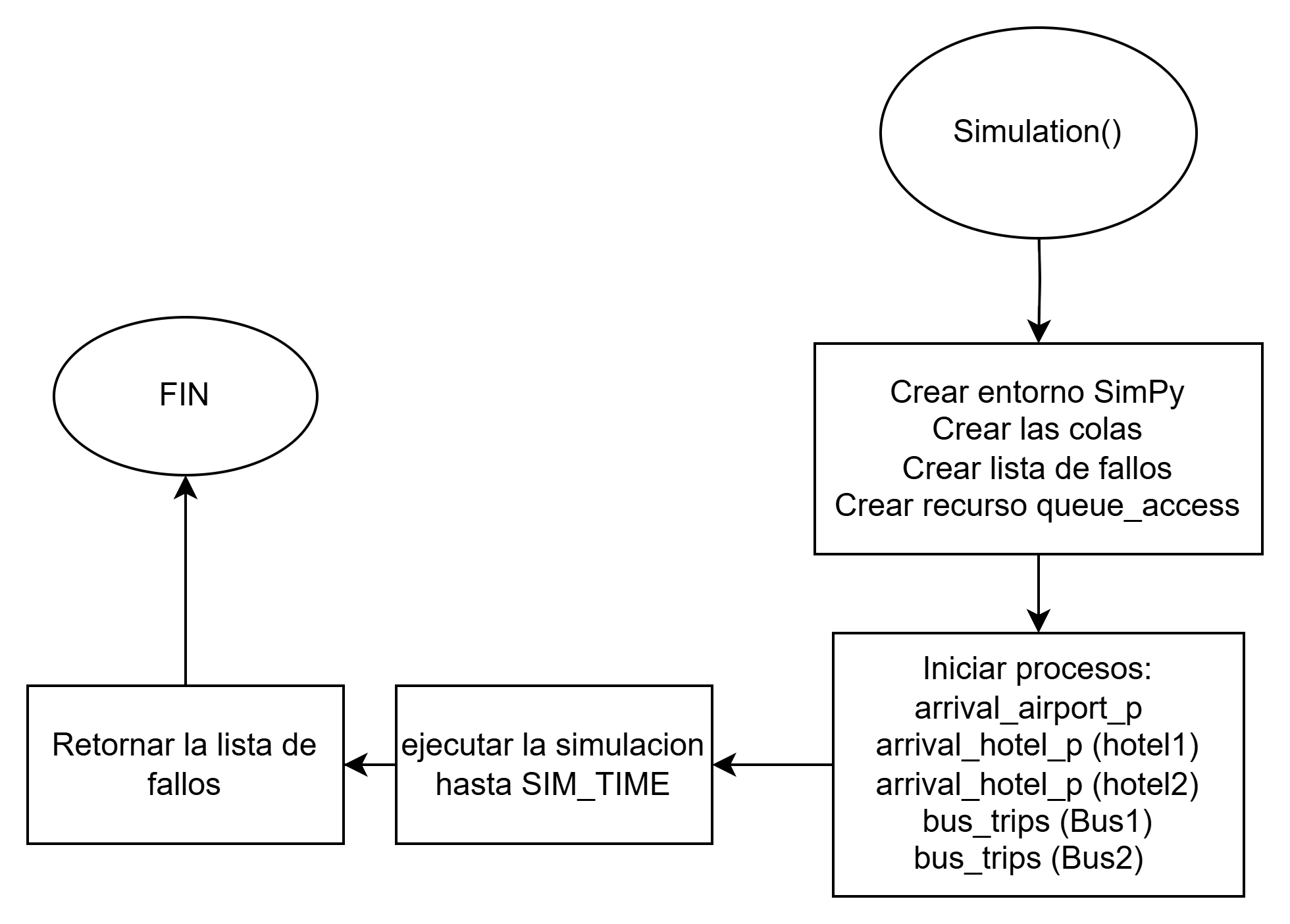
### **Resumen breve:** El código simula un día entero de trabajo (usamos 24 horas, ya que el ejercicio no especifica el tiempo) donde dos buses viajan entre un aeropuerto y dos hoteles, recogiendo y dejando pasajeros en las estaciones de cada destino. Lo que buscamos con esta simulación es encontrar el mínimo de capacidad que deberían tener los dos buses para que cuando lleguen a una estación todos los pasajeros estén en la cola de la estación sean recogidos, es decir ningún pasajero debe quedarse en una estación por falta de capacidad del bus. Esto se hará teniendo en cuenta principalmente el tiempo de viaje de los buses entre estaciones, la distribución de llegada de pasajeros a cada una de las estaciones para la cola del bus y que el segundo bus inicia 30 minutos después del primero. Con todo esto el código realiza la simulación usando un enfoque por eventos como veremos en los diagramas de flujo.

### Diagramas de flujo de programa principal y subprogramas:

#### Capacidad minima (programa principal):

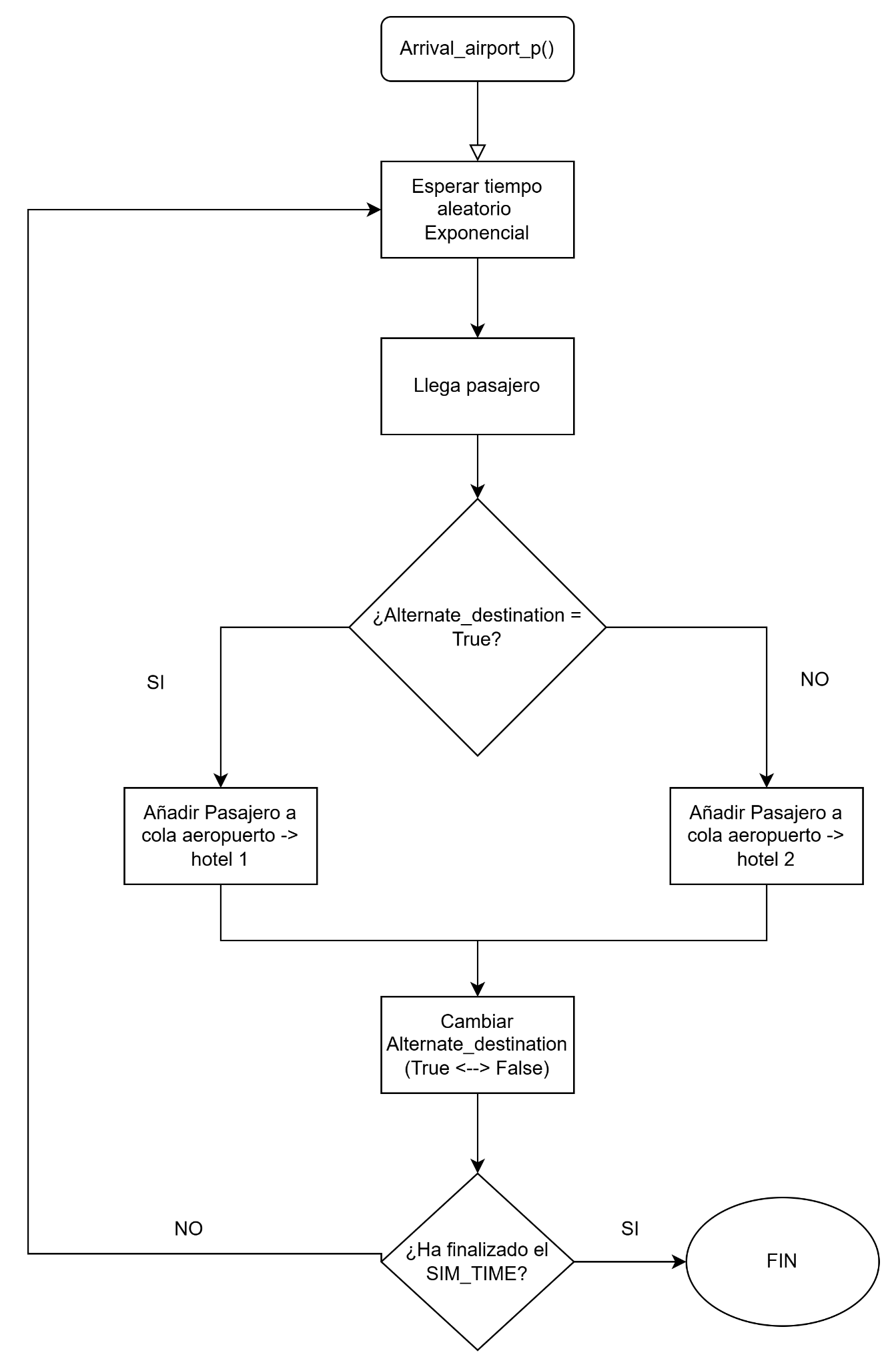


#### Simulation()

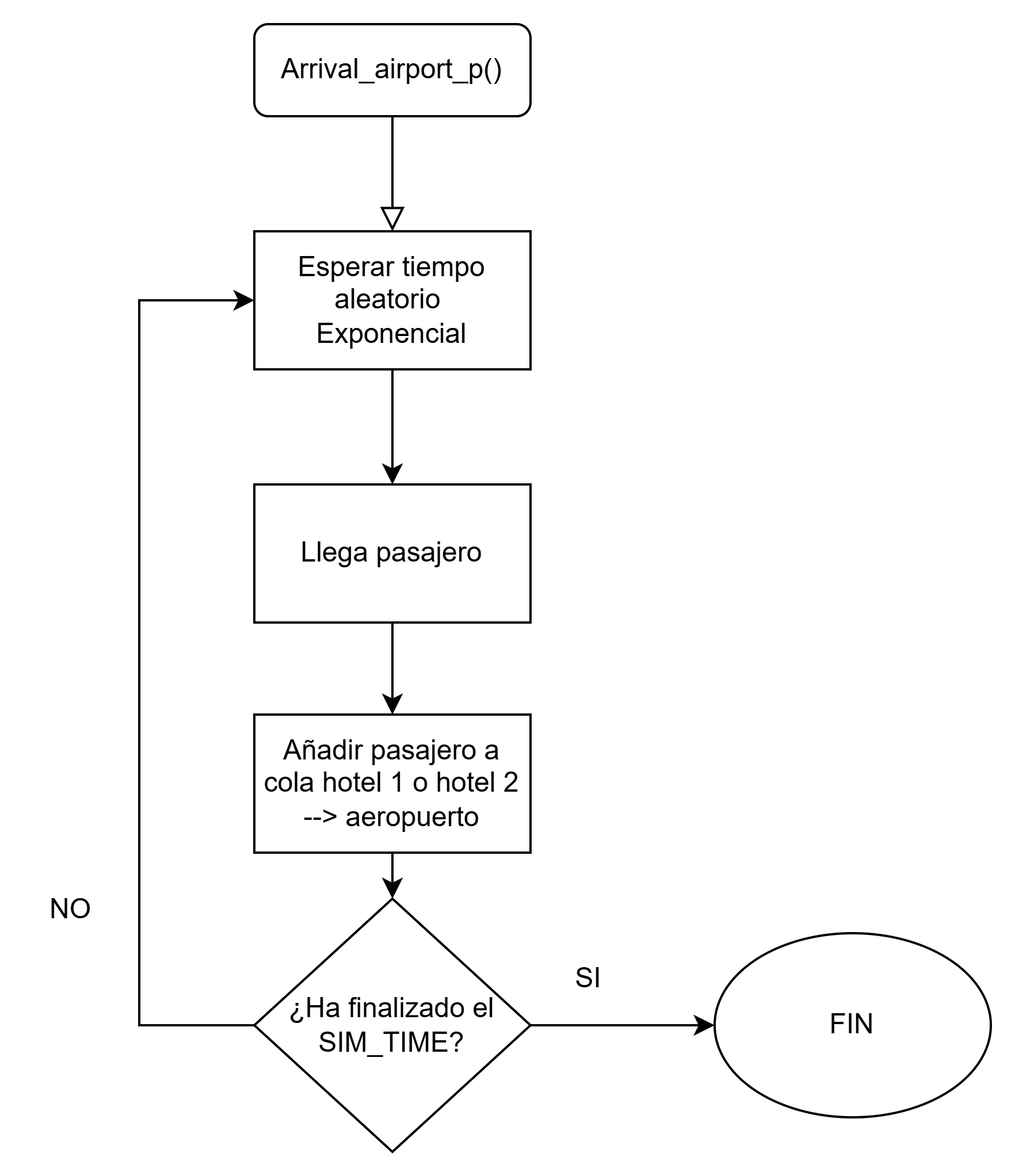
****

#### Subprogramas.

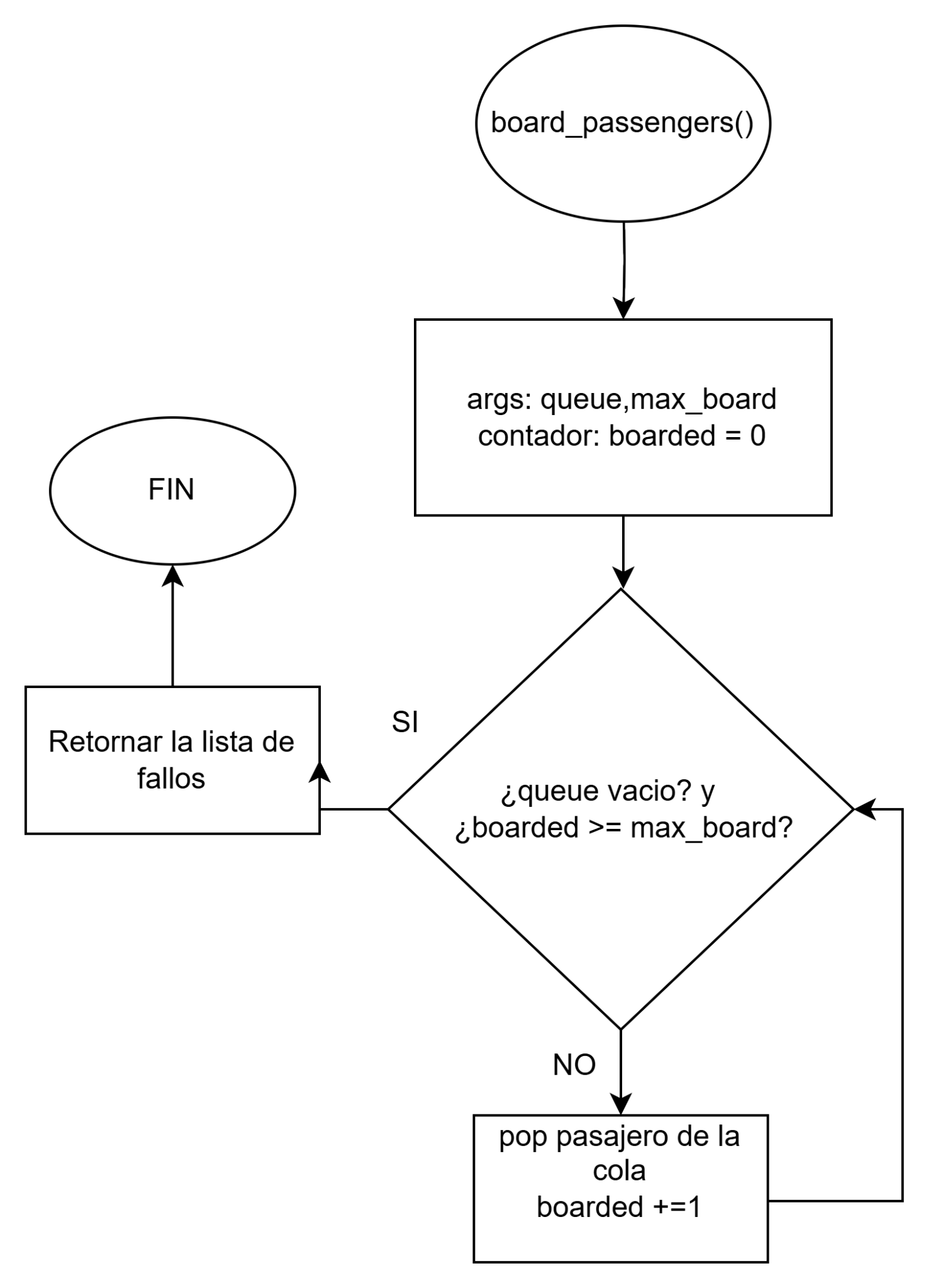
#### Pasajero llega a la estación del aeropuerto.



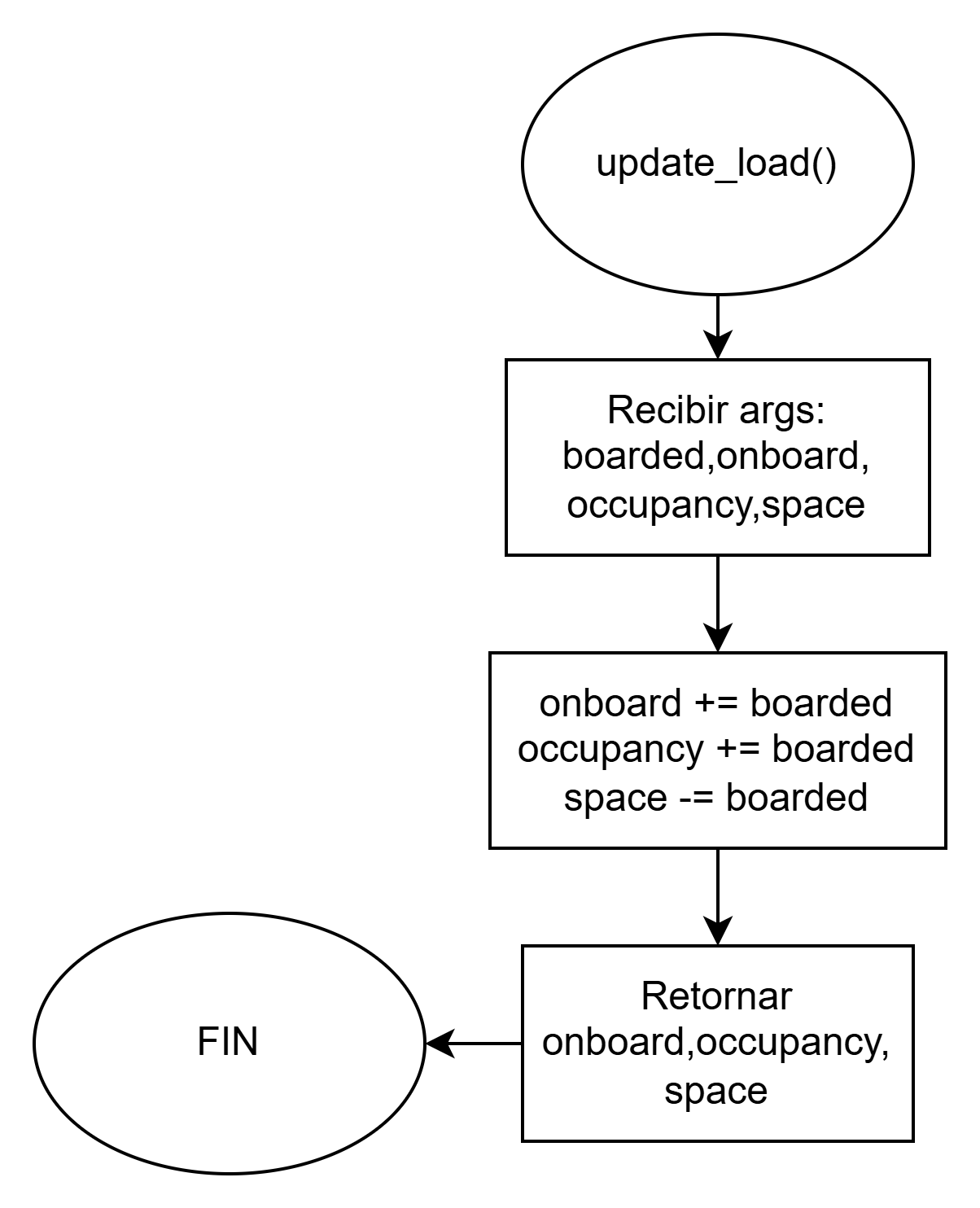
#### Pasajero llega a la estacion del hotel:



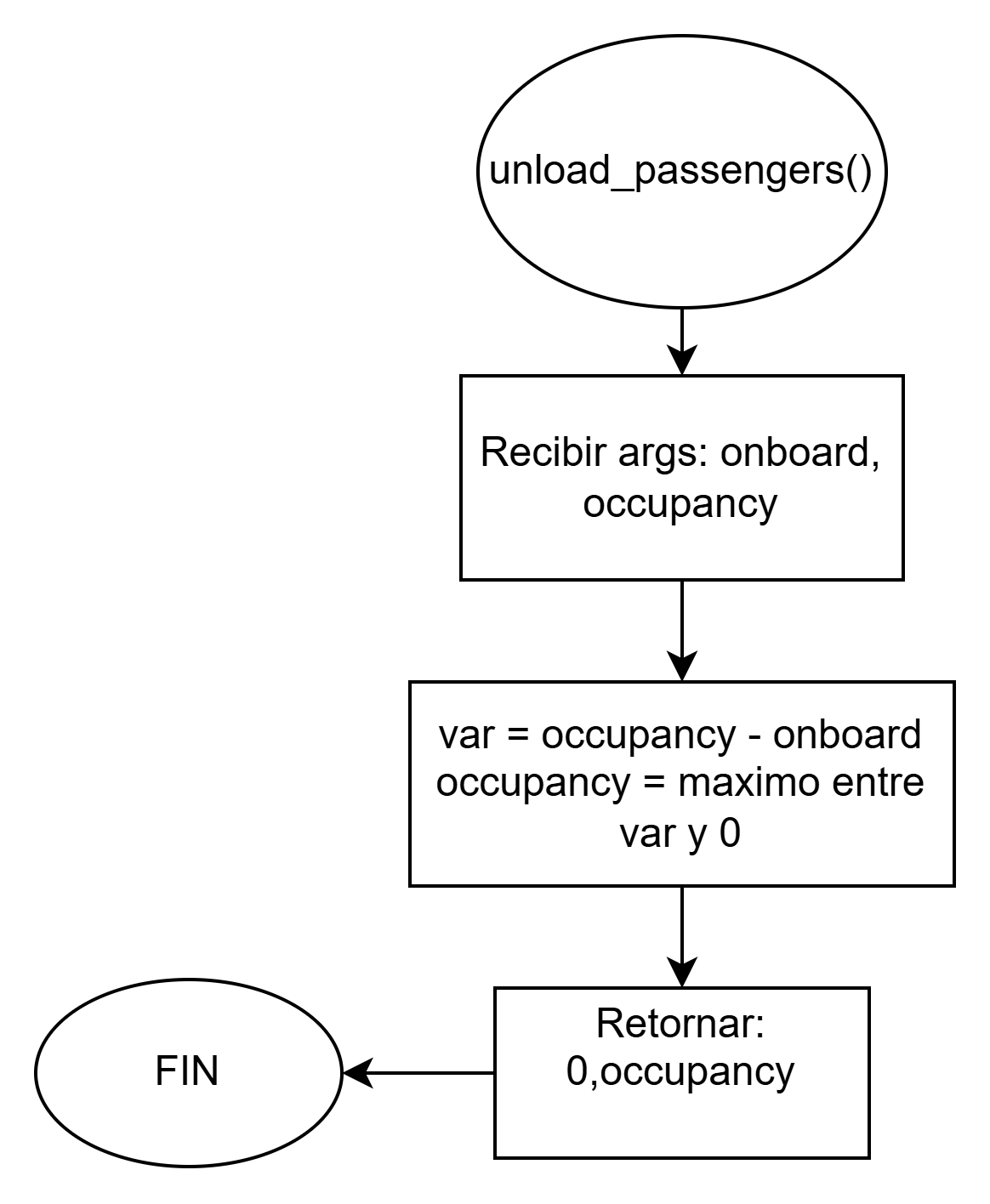
#### Pasajeros abordan el bus



#### Actualizar pasajeros a bordo



#### Descargar pasajeros



#### Viajes del bus



## Análisis de resultados:



* + El código hace un total de 5000 simulaciones con capacidades desde 10 pasajeros hasta 100, esto nos permite tener una mayor certeza del resultado que obtendremos ya que el sistema cuando algún bus no recoge la totalidad de pasajeros que hay en alguna cola, lo toma como falla y usando el acumulador de failures las vamos guardando. Así entonces vamos generando un porcentaje de éxito de las simulaciones con cada capacidad como se ve en la anterior imagen, hasta llegar a alguna capacidad que no haya fallado una sola vez. La simulación nos entrega como resultado que necesitamos que los buses tengan un mínimo de 44 asientos para pasajeros, de esta forma siempre que los buses lleguen a alguna estación al los pasajeros que se dirigían a esa estación quedará suficiente espacio para subir a los nuevos pasajeros, esto debido a que la tasa de llegada de los pasajeros a la estación del aeropuerto y a la del hotel es muy alta y los buses se demoran mucho más tiempo en llegar a cada estación. Además hay que tener en cuenta el tiempo que se aplica por la interferencia de procesos en cuanto a los buses principalmente. Entonces el número mínimo de pasajeros de 44 es un valor bastante aceptable y que tiene mucho sentido. Cabe aclarar que el ejercicio no ponía un tiempo límite de simulación así que la simulación se realizó teniendo en cuenta un dia entero de trabajo osea 24 horas continuas de gente llegando y los buses viajando si existiera un tiempo límite de trabajo como 8 horas el número de asientos mínimos se vería reducido a un valor entre 25 y 35.

## Modificaciones planteadas

* + Actualmente el simulador solo nos dice la capacidad mínima que necesitan los buses para que ningún pasajero se quede sin recoger, alguna modificación que a mi juicio mejorar el desempeño del sistema sería principalmente sincronizar las salidas de los buses, ya que actualmente el retraso del segundo y la salida rápida del primero ocasiona que buses salgan vacíos como se ve cuando hacemos la simulación, el primer momento no hay pasajeros ya que el bus 1 sale apenas inicia la simulación hacia el hotel 1, por lo tanto no se está aprovechando muy bien los buses.
  + También propongo mejorar el sistema de espera de los buses, actualmente recogen todas las personas de la cola, pero también debería agregar una lógica que espere un mínimo de personas para arrancar, o salir después de un tiempo máximo de espera, nuevamente con el fin de optimizar y mejorar la utilización de los buses.

# Ejercicio en SIMLIB

## Parámetros de entrada.

1. Tiempo de viaje de buses: el tiempo de viaje de los buses se distribuye normalmente con una *media* de 20 minutos y *desviación estándar* de 2 minutos, Los tiempos de viaje se cumplen para los trayectos: Aeropuerto → Hotel 1 → Hotel 2 → Aeropuerto.
2. Tiempo de llegada de los viajeros desde sus vuelos: El tiempo de viaje de los viajeros se distribuye exponencialmente con una *media* de 2,5 minutos. 50% de estos pasajeros se quedan en el Hotel 1 y el otro 50% se quedan en el Hotel 2
3. Tiempo llegada de personas al paradero de bus: En ambos hoteles cuando se bajan personas que llegan se suben otras que van hacia el aeropuerto, con un tiempo de llegada que se distribuye exponencialmente con una *media* de 5 minutos.
4. Tiempo de operación de los buses: El primer bus sale del aeropuerto al iniciar la simulación y el segundo bus sale 30 minutos después del primero.

## Variables del modelado.

* Capacidad del bus: Variable a determinar.
* Reloj de simulación (t = minutos transcurridos)
* Posición de los pasajeros.
* Posición del autobús:

## Descripción del evento y tipo de evento.

* Llegada pasajero aeropuerto: Un nuevo pasajero aterriza y se incrementa la variable de este contador. se programa según su distribución exponencial.
* Llegada pasajero parada bus Hotel 1: Un viajero quiere ir al aeropuerto y se incrementa la variable de este contador.
* Llegada pasajero parada bus Hotel 2: Un viajero quiere ir al aeropuerto y se incrementa la variable de este contador
* Llegada al Hotel 1: Descarga los pasajeros que van al Hotel 1, y se incrementa la variable del contador de pasajeros que van al aeropuerto.
* Llegada al Hotel 2: Descarga los pasajeros que van al Hotel 2, y e incrementa la variable del contador de pasajeros que van al aeropuerto
* Llegada al aeropuerto: El completa su viaje al llegar del Hotel 2 al aeropuerto, descarga a los pasajeros que querían ir al aeropuerto, y carga a los pasajeros que quieren ir a los hoteles.
* Fin de la simulación *(evento de control)* : Se alcanza un tiempo

## Listas y sus atributos

### Pasajeros:

* + Lugar de Origen (Aeropuerto, Hotel 1, Hotel 2)
  + Tiempo de llegada a las colas.
  + Lugar de destino (Aeropuerto, Hotel 1, Hotel 2)

### Bus:

* + Identificador de Bus (1 o 2)
  + Capacidad ( a definir)
  + Pasajeros a bordo.
  + Tiempo de viaje entre lugares.
  + Tiempo de inicio.

### Lugares:

* + Identificador de lugares.
  + Cola de pasajeros.

## Contadores y/o acumuladores:

### Contador de las colas de espera:

* + airport\_to\_hotel1, airport\_to\_hotel2: Pasajeros esperando en el aeropuerto
  + hotel1\_to\_airport,hotel2\_to\_airport: Pasajeros esperando en los hoteles.

### Estados del bus:

* + onboard\_H1, onboard\_H2, onboard\_A: Pasajeros a bordo por destino.
  + occupancy: Número total de pasajeros en el bus.
  + space: Número de espacios libres restantes.
  + boarded\_H1, boarded\_H2: pasajeros abordados para pasarlos a onboard

### Estados de simulación:

* + failures: Registro de eventos donde quedaron pasajeros sin transportar.
  + sim\_success: Conteo de simulaciones sin fallos.
  + rep: Número de repeticiones por configuración de capacidad.
  + capacity: Capacidad actual del bus en análisis.

## Medidas de desempeño:

### Success rate:

### Failures:

* + Cantidad de veces que algún bus no pudo recoger a todos los pasajeros en alguna estación. Se registra en una lista cada vez que hay más pasajeros en cola que espacios en el bus disponibles.

### Capacidad minima:

* + La capacidad mínima en los buses para que no ocurran fallos que se obtiene cuando el success rate es igual a la cantidad de repeticiones hechas en la simulación, osea cuando una capacidad al simularla no tuvo fallos.

## Subprogramas y propósito:

### void init\_simlib()

* **Propósito:** Inicializar la simulación.
* **Funcionamiento:**
  + Reinicia el tiempo de simulación a 0.
  + Vacía la lista de eventos.

### void event\_schedule()

* **Propósito:** Programar un evento futuro.
* **Funcionamiento:**
  + Añade un evento (tiempo y tipo) a la lista de eventos.
  + Verifica que no se exceda el número máximo (MAX\_EVENTS).

### void timing()

* **Propósito:** Obtener y ejecutar el próximo evento.
* **Funcionamiento:**
  + Busca el evento con menor tiempo (min\_time).
  + Actualiza sim\_time y event\_type.
  + Elimina ese evento de la lista.
* **Retorna:** Nada (pero actualiza variables globales).

### float expon(float mean, int stream)

* **Propósito:** Generar un número aleatorio con distribución **exponencial**.
* **Funcionamiento:**
  + Usa el método de inversión para generar tiempos entre llegadas.
  + No usa stream, está solo como parámetro simbólico.
* **Retorna:** Un número aleatorio exponencial con media mean.

### float normal()

* **Propósito:** Generar un número aleatorio con distribución **normal (gaussiana)**.
* **Funcionamiento:**
  + Usa el método de Box-Muller.
  + Guarda un valor para usarlo en la siguiente llamada (optimización).
* **Retorna:** Un número aleatorio normal con media mean y desviación estándar stddev.

### arrival\_airport\_p():

* **Propósito**: Genera pasajeros que llegan al aeropuerto para ser transportados a los hoteles.
* **Funcionamiento**: Alterna entre las colas de Hotel 1 y Hotel 2, generando llegadas exponenciales con media de 2.5 minutos.

### arrival\_hotel\_p() 1 y 2:

* **Propósito**: Simula la llegada de pasajeros a un hotel que desean regresar al aeropuerto.
* **Funcionamiento**: Agrega pasajeros a la cola del hotel correspondiente usando un proceso exponencial (media = 5 minutos).

### main()

* **Propósito:** Encuentra la capacidad mínima del bus que asegura 0 fallas en todas las simulaciones.
* **Funcionamiento:** Prueba capacidades desde 10 hasta 100. Ejecuta 5000 simulaciones por capacidad.Imprime la tasa de éxito y la mínima capacidad sin fallas.Retorna: 0 al finalizar (éxito del programa).

### advance\_time(int id)

* **Propósito:** Simula el tiempo de viaje del bus entre paradas.
* **Funcionamiento:** Avanza el tiempo usando una distribución normal (media 20, std 2).
* **Retorna:** Nada (modifica sim\_time diretamente).

### bus\_trips()

* **Propósito**: Modelo principal que simula el ciclo completo de viaje de un bus:

1. Recolecta pasajeros en el aeropuerto
2. Visita Hotel 1 → Hotel 2
3. Regresa al aeropuerto con los pasajeros de los hoteles

* **Registra fallos** si hay más pasajeros esperando que espacio disponible.

### simulation()

* **Propósito**: Ejecuta una instancia de simulación con una capacidad específica para los buses. Este es realmente el programa inicial.
* **Inicia** todos los procesos de llegadas y viajes.
* **Retorna**: Lista de fallos registrados.

### minimum\_capacity()

* **Propósito**: Encuentra la capacidad mínima del bus que garantice que no haya fallos en todas las repeticiones.
* **Repite** la simulación con diferentes capacidades hasta encontrar la óptima.

## Desarrollo en SIMLIB.

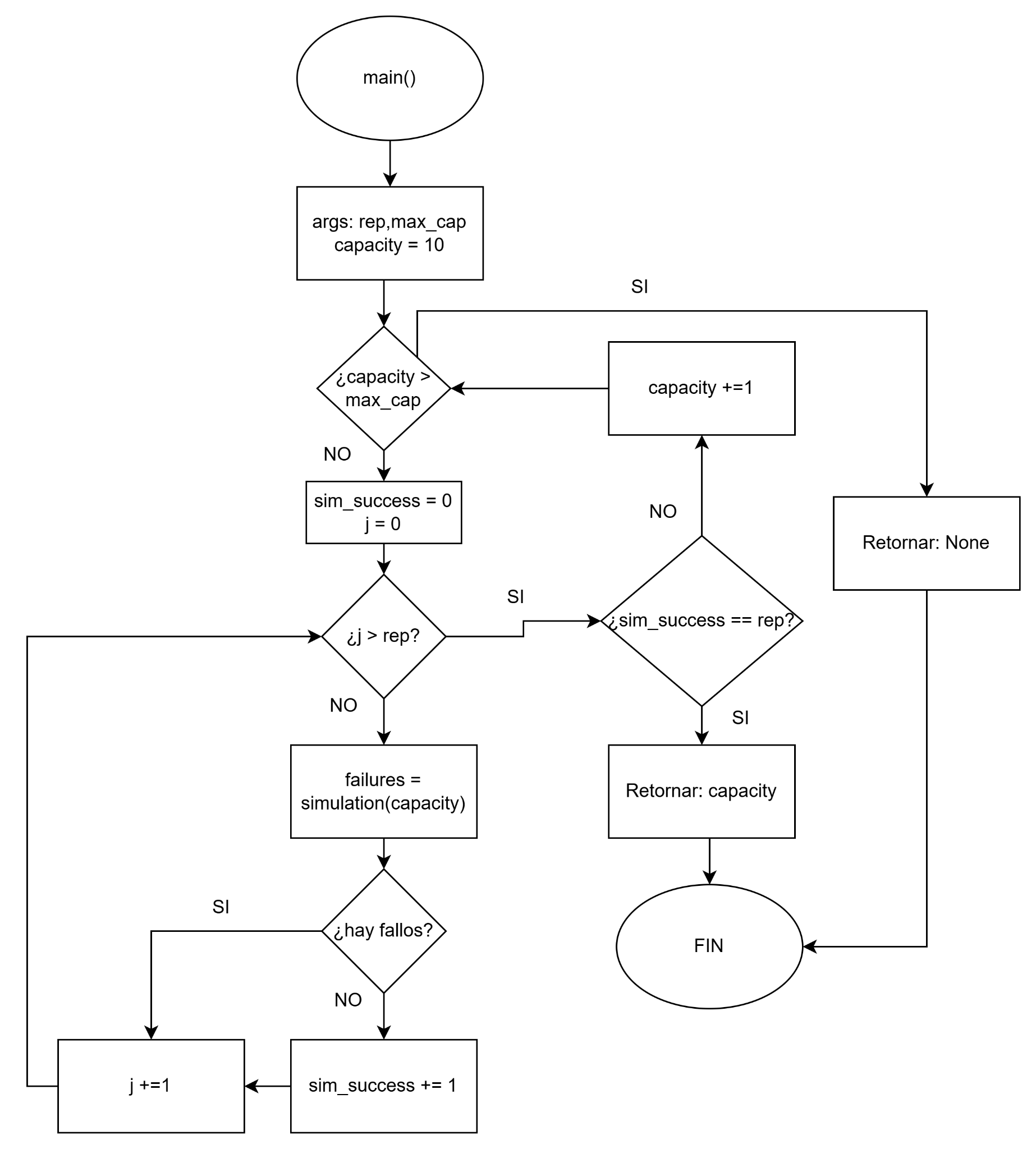
### Enlace al código: [Ejercicio 2.6.ipynb](https://colab.research.google.com/drive/1wB6KXpqjLmDQMqBYORGv0T2xq1pCSmLS#scrollTo=9CucYxMUkS19&line=7&uniqifier=1)

### Resumen breve: El código simula un día entero de trabajo (usamos 24 horas, ya que el ejercicio no especifica el tiempo) donde dos buses viajan entre un aeropuerto y dos hoteles, recogiendo y dejando pasajeros en las estaciones de cada destino. Lo que buscamos con esta simulación es encontrar el mínimo de capacidad que deberían tener los dos buses para que cuando lleguen a una estación todos los pasajeros estén en la cola de la estación sean recogidos, es decir ningún pasajero debe quedarse en una estación por falta de capacidad del bus. Esto se hará teniendo en cuenta principalmente el tiempo de viaje de los buses entre estaciones, la distribución de llegada de pasajeros a cada una de las estaciones para la cola del bus y que el segundo bus inicia 30 minutos después del primero. Con todo esto el código realiza la simulación usando un enfoque por eventos como veremos en los diagramas de flujo.

### 

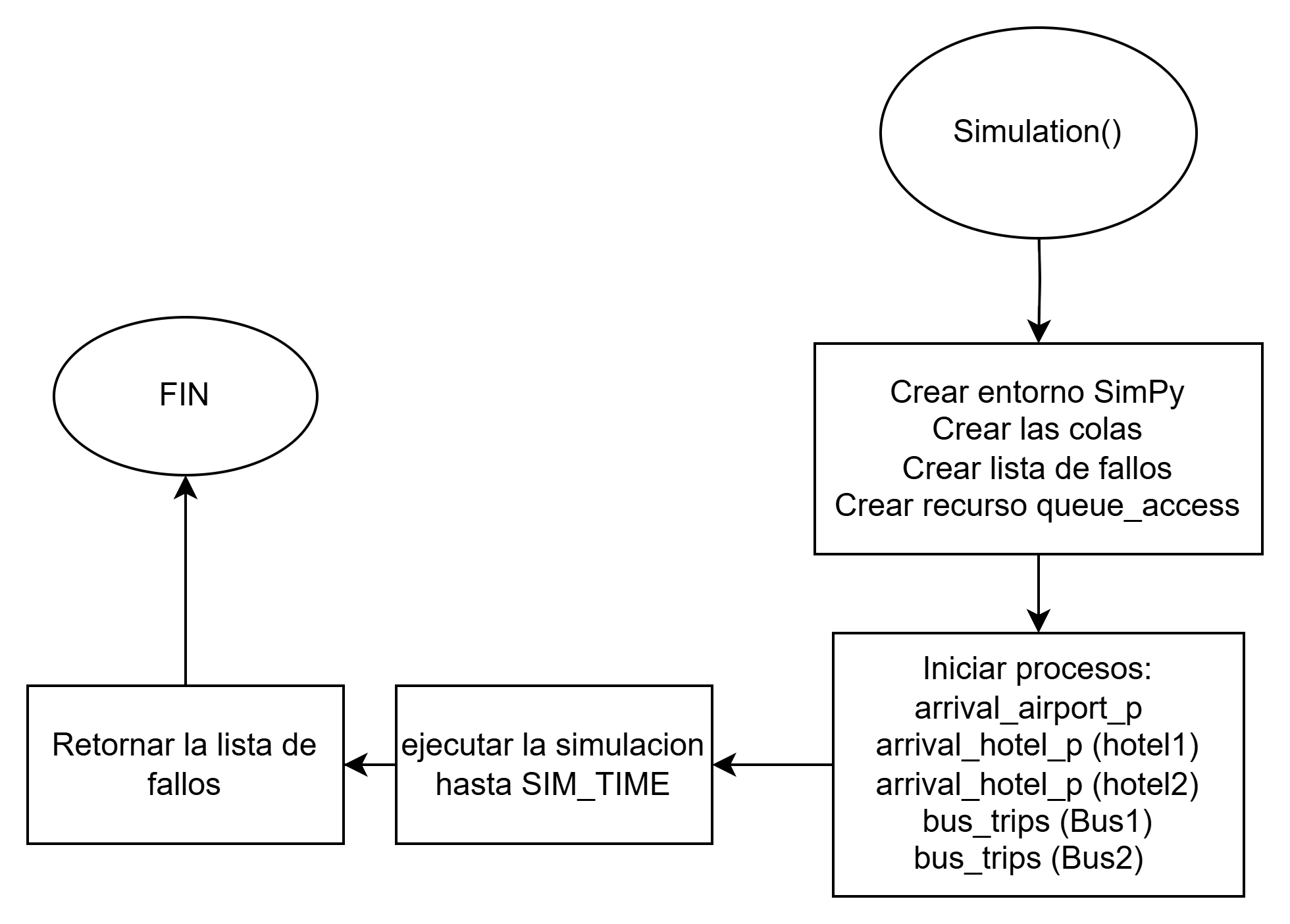
### Diagramas de flujo de programa principal y subprogramas:

#### main(): programa principal

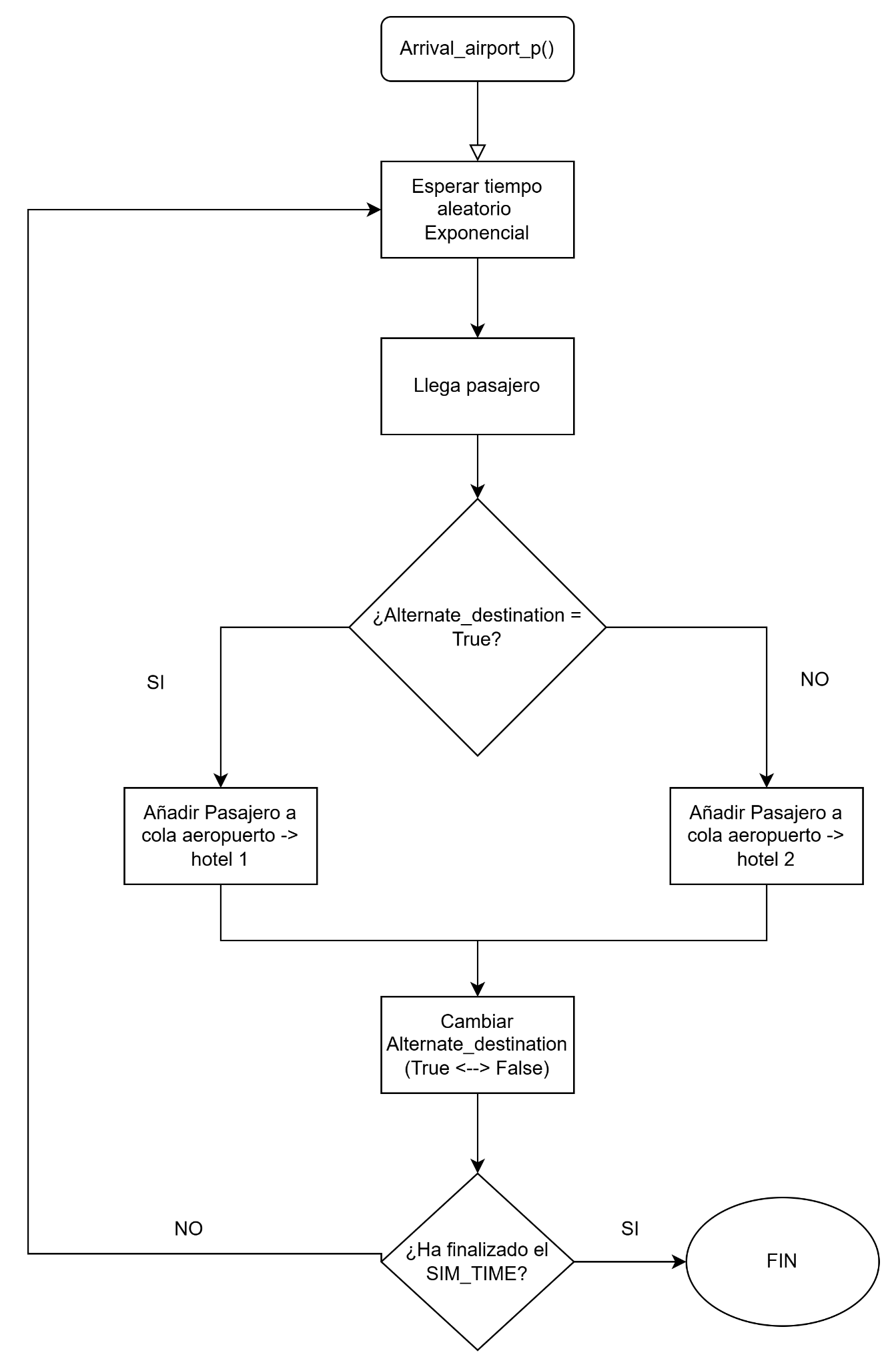


#### Subprogramas.

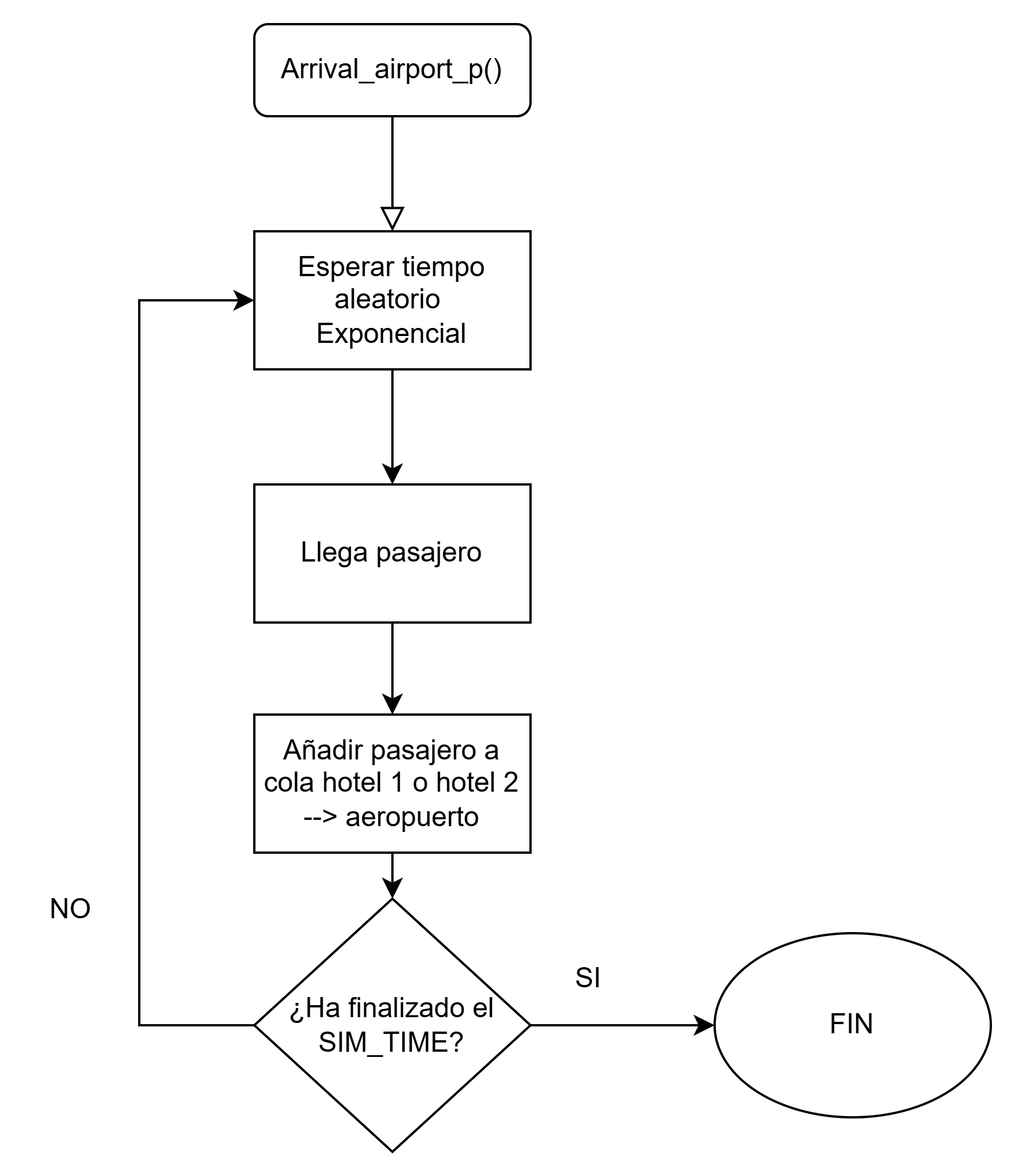
#### Simulation(): en vez de crear el entorno en simpy se crea en SIMLIB.

****

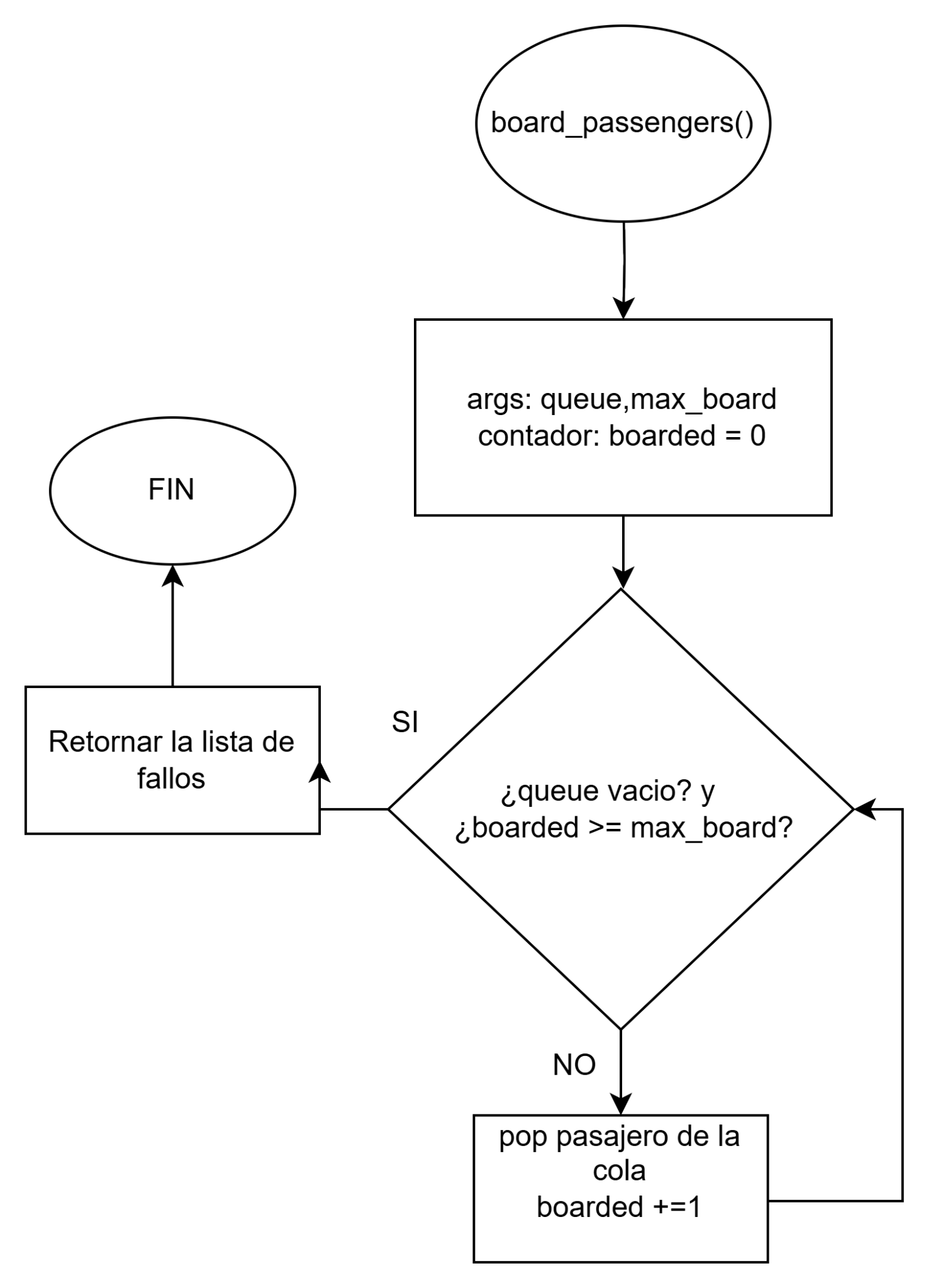
#### Pasajero llega a la estación del aeropuerto.



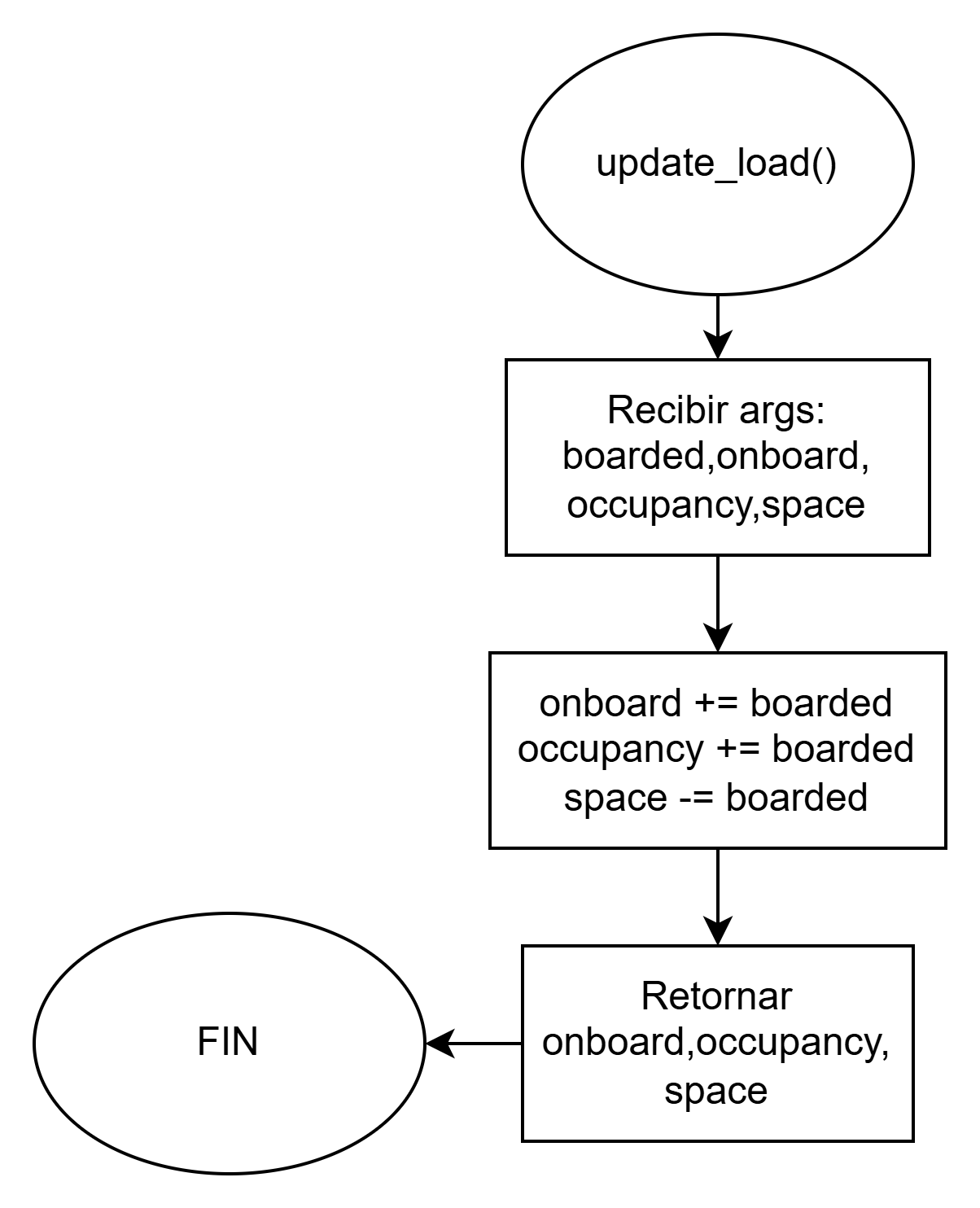
#### Pasajero llega a la estacion del hotel:



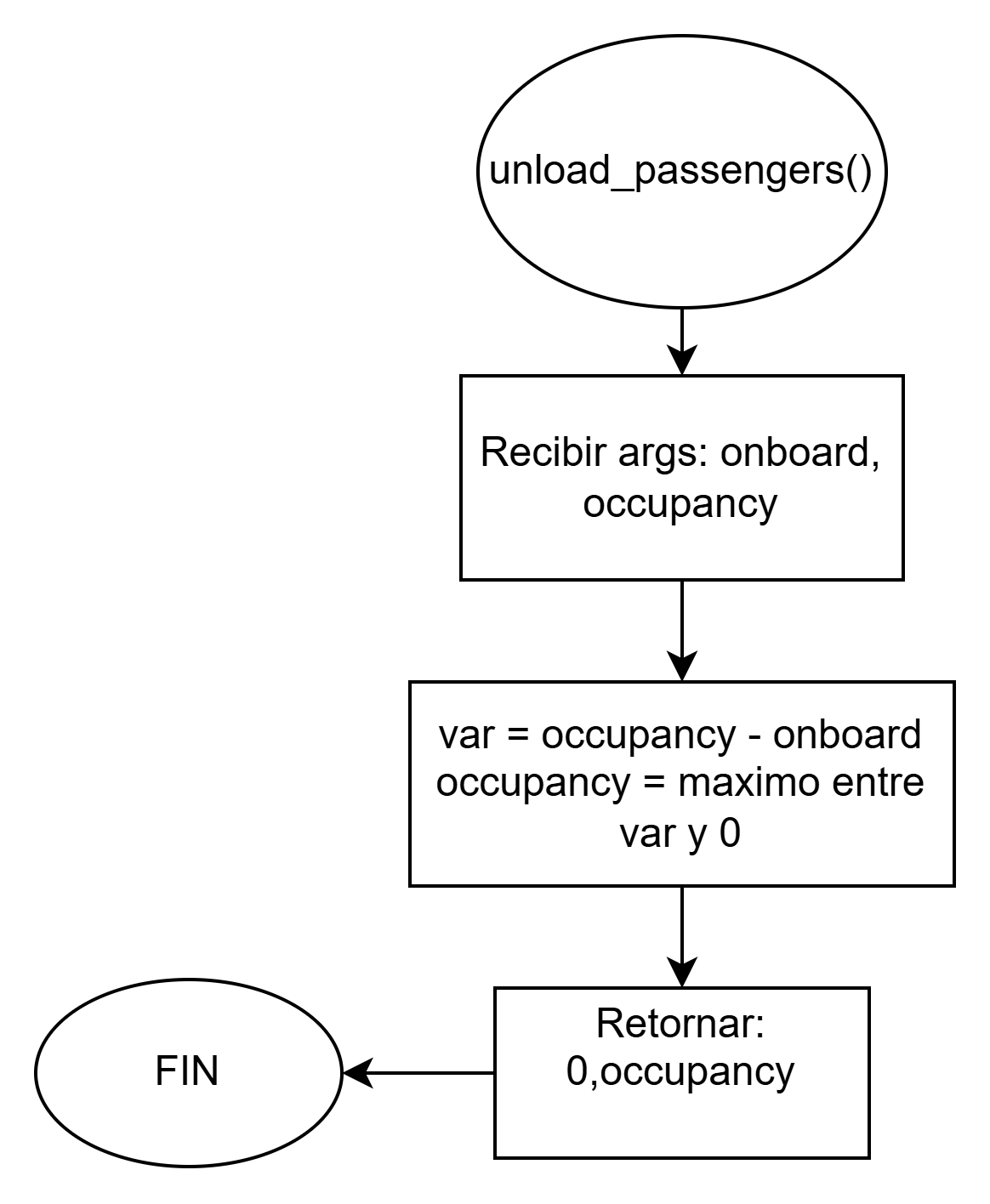
#### Pasajeros abordan el bus(subprograma implícito)



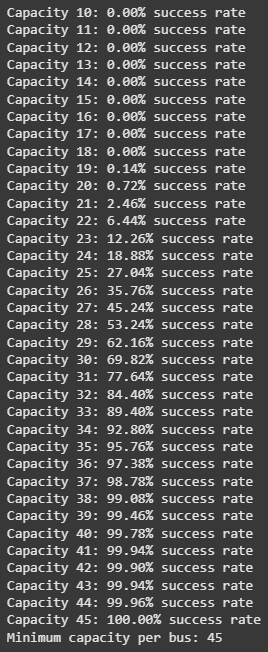
#### Actualizar pasajeros a bordo (subprograma implícito)



#### Descargar pasajeros(subprograma implícito)



## Análisis de resultados:



* + Al igual que el código en SimPy, definimos un total de 5000 simulación con capacidades desde 10 pasajeros hasta un máximo de 100, haciendo esto en ambos códigos podremos comparar cómo se ejecuta en cada uno y si obtenemos los mismos resultados, ya que la lógica que implementamos es muy similar, solo cambiamos algunas estructuras de datos por contadores que al final cumplen una función similar y no debería afectar ni sesgar los resultados de la simulación. Nuevamente generamos porcentajes de éxito por cada capacidad probada hasta hallar la capacidad más pequeña que tenga un 100% de éxito en las simulaciones, es decir no existieron fallos por lo tanto los buses con dicha capacidad no dejaron pasajeros en ningún momento esperando en las colas por falta de espacio. En este caso obtuvimos una capacidad mínima de 45 pasajeros por bus, nuevamente un valor bastante alto en general, y no muy lejos del obtenido en SimPy que fue 44, haciendo varias pruebas estos valores se mantienen entre 40 y 46, de pronto por la diferencia en algunas estructuras o como se manejan los eventos en SimPy con la interferencia de datos pueden variar 1 unidad o dos estos valores, pero sigue teniendo sentido debido a las altas tasas de llegada de los pasajeros a las diferentes estaciones que al compararlos con el tiempo de viaje que se demoran los buses con la distribución normal, los pasajeros se acumulan en las colas y a medida que se van viajando y llegando a las estaciones puede pasar que suban más pasajeros de los que se bajan por lo que se necesita tener una buena capacidad en los buses.

## Modificaciones planteadas

* + Como modificaciones principalmente recomendaría agregar estadísticas más completas, el código cumple igual que el de SimPy con el objetivo principal, pero considero que por temas de escalabilidad y usabilidad, sería necesario obtener estadísticas como cuanto se usaron los buses en promedio, propondría una refactorización del código y reorganización ya que podría mejorarse en términos de manejos de estructuras y posiblemente mejorar el tiempo que se demora simulando y manteniendo legibilidad, encapsulamiento de ciertas funciones como se realizó en SimPy en este código se repite unas partes de código que se podrían encapsular.
  + También se puede mejorar el manejo de eventos implementando una priority queue para evitar remover manualmente algunos elementos en cada evento.

# Referencias

1. A. M. Law, *Simulation Modeling and Analysis*, 5th ed. New York, NY, USA: McGraw-Hill, 2014. Código fuente de SIMLIB/C adaptado por ChatGPT (Gemini) para el propósito de la tarea, 25 de mayo de 2025.
2. ***Documentation for SimPy —*** *SimPy 4.1.2.dev8+g81c7218 documentation*. (s. f.). <https://simpy.readthedocs.io/en/latest/contents.html>
3. Jain, B. (2025, 5 marzo). *Simulating a Queuing System in Python*. Towards Data Science. <https://towardsdatascience.com/simulating-a-queuing-system-in-python-8a7d1151d485/>
4. Naps. (2018, 4 julio). *Simulación en Python usando Simpy: llegadas y servicio -*. Naps Tecnología y Educación. <https://naps.com.mx/blog/simulacion-en-python-usando-simpy/>