Proveedor 2 - Pizzita computing

```
!pip install simpy
```

```
Requirement already satisfied: simpy in /usr/local/lib/python3.10/dist-packages (4.0.2)
```

```
import simpy
import random
```

Task 1

```
import simpy
import random

# Parámetros iniciales del sistema
num_servers = 10  # Número de servidores
total_sim_time = 3600  # 1 hora en segundos
arrival_rate = 2400 / 60  # Tasa de llegada de solicitudes por segundo
service_rate = 1 / 600  # Tasa de servicio de solicitudes por segundo

# Listas para recopilar datos
tiempos_atendidos = []
tiempos_ocupados = []
tiempos_desocupados = []
tiempos_en_cola = []

# Definir un generador de llegada de solicitudes
def arrival(env, servers, arrival_rate, service_rate):
    i = 0
    while True:
        yield env.timeout(random.expovariate(arrival_rate))
        i += 1
        env.process(request(env, f'Solicitud {i}', servers, service_rate))

# Definir el proceso de solicitud
def request(env, name, servers, service_rate):
    global tiempos_atendidos, tiempos_en_cola
    with servers.request() as req:
        start = env.now
        yield req
        tiempos_atendidos.append(env.now - start)
        yield env.timeout(random.expovariate(service_rate))
        tiempos_en_cola.append(len(servers.queue))
        tiempos_ocupados.append(env.now - start)
        tiempos_desocupados.append(start - env.now)

# Inicializar el entorno de simulación
env = simpy.Environment()
servers = simpy.Resource(env, num_servers)

# Iniciar la simulación
env.process(arrival(env, servers, arrival_rate, service_rate))
env.run(until=total_sim_time)

# Respuestas a las preguntas de la tarea 1
print(f'a. Cantidad de solicitudes atendidas por cada servidor: {len(tiempos_atendidos) / num_servers}')
print(f'b. Tiempo total en que estuvo cada servidor ocupado: {sum(tiempos_ocupados)} segundos')
print(f'c. Tiempo total en que estuvo cada servidor desocupado: {sum(tiempos_desocupados)} segundos')
print(f'd. Tiempo total en que las solicitudes estuvieron en cola: {sum(tiempos_en_cola)} segundos')
print(f'e. Promedio de tiempo en cola por solicitud: {sum(tiempos_en_cola) / len(tiempos_en_cola)} segundos')
print(f'f. Promedio de solicitudes en cola por segundo: {sum(tiempos_en_cola) / total_sim_time} solicitudes')
print(f'g. Momento de salida de la última solicitud: {max(tiempos_atendidos) + max(tiempos_ocupados)} segundos')
```

```
a. Cantidad de solicitudes atendidas por cada servidor: 8.0
b. Tiempo total en que estuvo cada servidor ocupado: 126246.46942291802 segundos
c. Tiempo total en que estuvo cada servidor desocupado: -126246.46942291802 segundos
d. Tiempo total en que las solicitudes estuvieron en cola: 5042215 segundos
e. Promedio de tiempo en cola por solicitud: 72031.64285714286 segundos
```

```
f. Promedio de solicitudes en cola por segundo: 1400.6152777777777 solicitudes
g. Momento de salida de la última solicitud: 7180.895526419169 segundos
```

Task 2

```
import simpy
import random
```

```
import simpy
import random

# Parámetros iniciales del sistema
num_servers = 1  # Número inicial de servidores
total_sim_time = 60 * 60  # 1 hora en segundos
arrival_rate = 2400 / 60  # Tasa de llegada de solicitudes por segundo
service_rate = 1 / 600  # Tasa de servicio de solicitudes por segundo
servers_needed = 0
busy_servers = 0

# Definir un generador de llegada de solicitudes
def arrival(env, servers, arrival_rate, service_rate):
    i = 0
    while True:
        yield env.timeout(random.expovariate(arrival_rate))
        i += 1
        env.process(request(env, f'Solicitud {i}', servers, service_rate))

# Definir el proceso de solicitud
def request(env, name, servers, service_rate):
    global servers_needed, busy_servers
    if 'servers_needed' not in globals():
        servers_needed = 0
    with servers.request() as req:
        if servers.count == busy_servers:
            servers_needed = max(servers_needed, servers.count + 1)
        busy_servers = servers.count
        yield req
        busy_servers -= 1
        yield env.timeout(random.expovariate(service_rate))

# Inicializar el entorno de simulación
env = simpy.Environment()
servers = simpy.Resource(env, num_servers)

# Iniciar la simulación
env.process(arrival(env, servers, arrival_rate, service_rate))
env.run(until=total_sim_time)

print(f"Se necesitan al menos {servers_needed} servidores para garantizar que siempre haya al menos un servidor disponible.")
```

```
Se necesitan al menos 2 servidores para garantizar que siempre haya al menos un servidor disponible.
```

Task 3

```
import simpy
import random

# Parámetros iniciales del sistema
num_servers = 10  # Número de servidores
total_sim_time = 3600  # 1 hora en segundos
arrival_rate_initial = 2400 / 60  # Tasa de llegada inicial de solicitudes por segundo
arrival_rate_new = 6000 / 60  # Nueva tasa de llegada de solicitudes por segundo
service_rate = 1 / 600  # Tasa de servicio de solicitudes por segundo
servers_needed = 0

# Definir un generador de llegada de solicitudes
def arrival(env, servers, arrival_rate, service_rate):
    i = 0
```

```
        while True:
            yield env.timeout(random.expovariate(arrival_rate))
            i += 1
            env.process(request(env, f'Solicitud {i}', servers, service_rate))

    # Definir el proceso de solicitud
    def request(env, name, servers, service_rate):
        with servers.request() as req:
            yield req
            yield env.timeout(random.expovariate(service_rate))

    # Inicializar el entorno de simulación para el nuevo caso
    env = simpy.Environment()
    servers_new = simpy.Resource(env, num_servers)
    wait_times = []
    queue_lengths = []

    # Función de solicitud para el nuevo caso
    def request_new(env, name, servers, service_rate):
        with servers.request() as req:
            arrival_time = env.now
            yield req
            wait_times.append(env.now - arrival_time)
            queue_lengths.append(len(servers.queue))
            yield env.timeout(random.expovariate(service_rate))

    # Proceso de llegada de solicitudes para el nuevo caso
    def arrival_new(env, servers, arrival_rate, service_rate):
        i = 0
        while True:
            yield env.timeout(random.expovariate(arrival_rate))
            i += 1
            env.process(request_new(env, f'Solicitud {i}', servers, service_rate))

    # Iniciar la simulación para el nuevo caso
    env.process(arrival_new(env, servers_new, arrival_rate_new, service_rate))
    env.run(until=total_sim_time)

    # Resultados para el nuevo caso
    # a. Cantidad de solicitudes atendidas por cada servidor
    solicitudes_atendidas_por_servidor = total_sim_time / service_rate / num_servers
    print(f"Solicitudes atendidas por cada servidor: {solicitudes_atendidas_por_servidor}")

    # b. Tiempo total en que estuvo cada servidor ocupado
    tiempo_total_ocupado_por_servidor = sum(wait_times) + total_sim_time / num_servers
    print(f"Tiempo total en que estuvo cada servidor ocupado: {tiempo_total_ocupado_por_servidor}")

    # c. Tiempo total en que estuvo cada servidor desocupado (idle)
    tiempo_total_desocupado_por_servidor = total_sim_time - tiempo_total_ocupado_por_servidor
    print(f"Tiempo total en que estuvo cada servidor desocupado: {tiempo_total_desocupado_por_servidor}")

    # d. Tiempo total en que las solicitudes estuvieron en cola
    tiempo_total_en_cola = sum(wait_times)
    print(f"Tiempo total en que las solicitudes estuvieron en cola: {tiempo_total_en_cola}")

    # e. Promedio de tiempo en cola por solicitud
    promedio_tiempo_en_cola = tiempo_total_en_cola / len(wait_times)
    print(f"Promedio de tiempo en cola por solicitud: {promedio_tiempo_en_cola}")

    # f. Promedio de solicitudes en cola por segundo
    promedio_solicitudes_en_cola_por_segundo = sum(queue_lengths) / total_sim_time
    print(f"Promedio de solicitudes en cola por segundo: {promedio_solicitudes_en_cola_por_segundo}")

    # g. Momento de salida de la última solicitud
    momento_salida_ultima_solicitud = max(wait_times) + total_sim_time
    print(f"Momento de salida de la última solicitud: {momento_salida_ultima_solicitud}")
```

```
Solicitudes atendidas por cada servidor: 216000.0
Tiempo total en que estuvo cada servidor ocupado: 104751.98266795259
Tiempo total en que estuvo cada servidor desocupado: -101151.98266795259
Tiempo total en que las solicitudes estuvieron en cola: 104391.98266795259
Promedio de tiempo en cola por solicitud: 1470.3096150415856
Promedio de solicitudes en cola por segundo: 2907.5375
Momento de salida de la última solicitud: 7101.735293471835
```

```
# Tarea 2: Determinar la cantidad de servidores necesarios para garantizar que siempre haya al menos un servidor disponible
servers_needed = 0
total_sim_time = 2 * 3600  # 2 horas en segundos, o cualquier valor mayor si es necesario

def check_servers_new(env, servers, arrival_rate, service_rate):
    global servers_needed
    while True:
        yield env.timeout(1)  # Verificar cada segundo
        if servers.count < 1:
            servers_needed = max(servers_needed, 1)

# Iniciar la simulación para el nuevo caso
env.process(arrival_new(env, servers_new, arrival_rate_new, service_rate))
env.process(check_servers_new(env, servers_new, arrival_rate_new, service_rate))
env.run(until=total_sim_time)

print(f"Se necesitan al menos {servers_needed} servidores para garantizar que siempre haya al menos un servidor disponible.")
```

```
Se necesitan al menos 0 servidores para garantizar que siempre haya al menos un servidor disponible.
```