Dependencias de Estado

PROG. CONCURRENTE Y DE TIEMPO REAL

UNIVERSIDAD DE BURGOS

Jose Maria Santos Romero

Índice

Descripción del programa

Esta práctica trata de solucionar el **problema de exclusión mutua** de un parque con entradas y salidas. Por lo que este programa simula el flujo de entradas y salidas de un parque durante un periodo de tiempo determinado de ejecución hasta que se alcancen 20 entradas y sus correspondientes 20 salidas, en este caso por 5 puertas. Además el aforo máximo del parque son 50 personas.

El sistema lanzador recibe un único parámetro de argumento, este debe ser el número de puertas que tiene el parque y este creará tantos pares de hilos como puertas haya, en este caso como estamos lanzando la simulación con 5 puertas, de la 'A' a la 'E' el programa lanzará un total de 10 tareas, 5 para entradas y 5 para salidas.

Tabla de Dependencias de Estado

Consideramos tres estados basados en el contador de personas dentro del parque, que son Superior, Medio e Inferior.

Superior: Estado en el que sólo es posible la transición hacia el estado medio al decrementar el valor o inferior si el estado intermedio no existiera (*parque lleno*).
Medio: Estado en el que es posible tanto incrementar el valor y pasar al estado superior o decrementar el valor y pasar al estado inferior (*parque con espacio*).
Inferior: Estado en el que sólo es posible la transición hacia el estado medio al incrementar el valor o superior si el estado intermedio no existiera (*parque vacío*).

Estado	Condición	`entrarAlParque()`	`salirDelParque()`
Superior	`contadorPersonasTotales == maxPersonas`	**No**	**Si**
Medio	`0<contadorPersonasTotales<maxPersonas`	**Si**	**Si**
Inferior	`contadorPersonasTotales == 0`	**Si**	**No**

Explicación de ejecución

A continuación se muestra un fragmento de la salida por pantalla tras la ejecución de la simulación del flujo de entradas y salidas de personas al parque con cinco puertas.

```
¡Parque abierto!
Entrada por puerta A
-> Personas en el parque 1
--> Por puerta A 1

Salida por puerta A
-> Personas en el parque 0
--> Por puerta A 0

Entrada por puerta C
-> Personas en el parque 1
--> Por puerta A 0
--> Por puerta C 1

Salida por puerta B
-> Personas en el parque 0
--> Por puerta A 0
--> Por puerta C 1
--> Por puerta B -1
```

En la primera línea vemos un mensaje de bienvenida tras inicializar el parque.
A continuación, tras haber lanzado las 10 tareas, 5 entradas y 5 salidas…
1.- Entra la primera persona al parque por la puerta A, por lo que el contador de personas en el parque es 1 y el contador por la puerta A es 1.
2.- Sale la persona que estaba en el parque por la puerta A, por lo que el contador de la puerta A ha sido decrementado a 0 y ahora las personas en el parque es igual a 0.
3.- Entra una persona por la puerta C, por lo que el contador de personas en el parque es 1 y el contador por la puerta C es 1.
4.- Sale la persona que estaba en el parque por la puerta B, que todavía no había entrado nadie por ella, por lo que decrementamos su contador de 0 a -1, y el contador de la puerta C sigue siendo 1 pero las personas en el parque son 0 porque ha salido por la puerta B.

Permitimos que el contador de las puertas puedan tener valores negativos para no perder nada de información en el flujo del tráfico, ya que por ejemplo si una persona entra por la puerta C y sale por la puerta B por la que todavía no ha entrado nadie, y el contador no permitiera valor negativo, no sabrías por qué puerta ha salido la persona que estaba en el parque, ya que el contador de la puerta B te mostraría 0.

Por lo que esta implementación es necesaria para no perder la información de por qué puerta salió la persona y llevar así un seguimiento en tiempo real.