Universidad Nacional Autónoma de México

Sistemas Operativos

Una Situación Cotidiana Paralelizable

Grupo 6

Nombre:
Barreiro Valdez Alejandro
Zepeda Baeza Jessica

Proyecto 2

Profesor: Ing. Gunnar Eyal Wolf Iszaevich

31 de marzo de 2022

1. Descripción

La situación que se planteará en este proyecto sale de los mecanismos en la espera en el abordaje de un metrobús. El problema consta de que muchas personas pueden entrar a un metrobús pero solo cuando este llega, mientas esto pasa se tiene una espera indefinida de personas en la estación. Se busca paralelizar la entrada de las personas al metrobús y cómo este transporta a muchas personas. Sin embargo, existen varios eventos a controlar para lograr modelar este problema como en la vida real. En este caso se trata a los pasajeros y a los metrobuses como hilos independientes que necesitan sincronización, es decir, varios hilos de personas se asocian a un metrobús.

2. Eventos a controlar

Se debe regular cuántas personas entran ya que el metrobús tiene cupo y no pueden entrar todas. En el caso de este problema solo pueden entrar 10 personas a cada metrobuses. Una vez que las personas hayan abordado el metrobús debe partir de la estación. Además, si no hay pasajeros esperando al metrobús entonces éste se seguirá de largo para optimizar el tiempo. Pueden llegar dos metrobuses al mismo tiempo y estos se pueden repartir la gente que está esperando.

3. Ordenamiento relativo

En este problema en particular el orden relativo no afecta y no es de importancia para la solución. En el caso real de lo que se plantea no importa el orden en que las personas ingresan al metrobús y tampoco importa el hecho de que un hilo sea el que menos espera para abordar el metrobús y aún así aborde antes.

4. Mecanismos de sincronización

4.1. Mutex

Se emplea un Mutex para proteger las aquellas variables que son modificadas por varios hilos como son los contadores de pasajeros esperando un metrobús. También se emplean para la obtención de los máximos y mínimos a lo largo del código ya que son variables almacenadas en memoria que no pueden ser accesadas y modificadas por varios hilos en un mismo instante.

4.2. Barrera

Se emplea dentro del concepto de metrobús para dejar pasar a todos los hilos pasajeros cuando llega un metrobús. La barrera implementada se inicializa con un valor calculado de los pasajeros que llegan; de manera que siempre se liberarán pasajeros diferentes dependiendo de los hilos que se creen. La principal ventaja de esto es que se tendrá una barrera exacta que no tenga que esperar a que llegue un cierto número de pasajeros (que probablemente nunca pase).

4.3. Semáforo

Se emplea como una variable que comunica los metrobuses y los pasajeros de manera que "espera" a que los pasajeros "entren" al metrobús.

5. Lógica de Operación

5.1. Estado compartido

En este programa la variable que estará siendo compartida y manipulada por una función metrobús y una función pasajeros será el número de personas que están esperando. Esta variable se declara como global en ambas funciones para poder acceder a ella.

5.2. Descripción algorítmica de cada hilo

El primer hilo que aparece en el programa es el del metrobús. Se adquiere un *mutex* para manipular la variable de personas esperando. Después se obtiene la cantidad de pasajeros que puede entrar al metrobús con una función mínima. Se crea una nueva barrera con ese número para las personas que ingresan al metrobús. Posteriormente, utilizando un semáforo se espera a que los pasajeros aborden y se calcula el nuevo número de personas esperando. Por último se libera el *mutex* y el metrobús parte de la estación.

El segundo tipo de hilo que se programó es el de pasajeros, este hilo adquiere el *mutex* para aumentar en uno la cantidad de personas que se encuentran esperando. Posteriormente, espera a la barrera que le puso el metrobús para poder abordar al metrobús. El abordo se realiza con un semáforo de abordo.

5.3. Interacción entre los hilos

La interacción entre hilos se encuentra en el semáforo de abordo y la barrera del metrobús que este pone. Estas dos variables permiten que los pasajeros puedan entrar a tiempo al metrobús y puedan esperar a su entrada.

6. Entorno de desarrollo

Se utilizó Python 3.6.8 para el desarrollo de este código. El código se ejecutó con esta versión de Python utilizando el comando *python3*. Adicionalmente, dentro del código para la interfaz de usuario, se utiliza el comando *clear*. Este comando se encuentra en sistemas operativos MacOS y GNU/Linux. Este programa fue probado en MacOS y en CentOS Stream 8. Si se ejecuta en otro sistema que no contenga este comando podría lanzar alguna excepción o error el código. Las bibliotecas que se importan en el código pertenecen a *The Python Standard Library* y tienen compatibilidad con la versión antes mencionada de Python. El programa es una pantalla que permite ver los movimientos de las personas y el metrobús. Se puede cambiar a la siguiente pantalla pulsando cualquier tecla y para terminar la ejecución se pulsa la tecla *q*.

7. Capturas de ejecuciones

Se generaron tres capturas de pantalla del código generado para demostrar su ejecución exitosa. En la Figura 1 se muestra una captura donde se muestra el número de pasajeros que han llegado y cuántos metrobuses han llegado. En este caso han llegado 15 personas y 2 metrobuses. Por lo que 10 abordarán el primero y el segundo solo 4. Las personas esperando en este caso son 0.

Para la Figura 2 se tiene que llegaron 13 pasajeros y solo un metrobús. En este caso el metrobús se irá lleno y solo esperarán 3 personas a abordar en el siguiente metrobús que llegue.

En la siguiente captura de pantalla de la Figura 3 llegan 5 pasajeros y dos metrobuses. Adicionalmente, existían tres personas de la captura anterior que estaban esperando por lo que dichas personas se suben al primer metrobús con las otras cinco que habían llegado para que sean 8. Para el segundo metrobús ya no hay gente. Después de esta pantalla se ingresó la tecla q y se terminó la ejecución del programa.

```
Llegada de 14 pasajero(s)
El pasajero está llegando. Esperando: 1
El pasajero está llegando. Esperando: 2
El pasajero está llegando. Esperando: 3
El pasajero está llegando. Esperando: 4
El pasajero está llegando. Esperando: 5
El pasajero está llegando. Esperando: 6
El pasajero está llegando. Esperando: 7
El pasajero está llegando. Esperando: 8
El pasajero está llegando. Esperando: 9
El pasajero está llegando. Esperando: 10
El pasajero está llegando. Esperando: 11
El pasajero está llegando. Esperando: 12
El pasajero está llegando. Esperando: 13
El pasajero está llegando. Esperando: 14
Llegada de 2 metrobus
El metrobús ha partido. Pasajeros: 10
El metrobús ha partido. Pasajeros: 4
Personas esperando: 0
Pulsar enter para continuar. Ingresar q para salir:
```

Figura 1: Captura de ejecución exitosa

```
Llegada de 13 pasajero(s)
El pasajero está llegando. Esperando: 1
El pasajero está llegando. Esperando: 2
El pasajero está llegando. Esperando: 3
El pasajero está llegando. Esperando: 4
El pasajero está llegando. Esperando: 5
El pasajero está llegando. Esperando: 6
El pasajero está llegando. Esperando: 7
El pasajero está llegando. Esperando: 8
El pasajero está llegando. Esperando: 9
El pasajero está llegando. Esperando: 10
El pasajero está llegando. Esperando: 11
El pasajero está llegando. Esperando: 12
El pasajero está llegando. Esperando: 13
Llegada de 1 metrobus
El metrobús ha partido. Pasajeros: 10
Personas esperando: 3
Pulsar enter para continuar. Ingresar q para salir:
```

Figura 2: Captura de ejecución exitosa

```
Llegada de 5 pasajero(s)
El pasajero está llegando. Esperando: 4
El pasajero está llegando. Esperando: 5
El pasajero está llegando. Esperando: 6
El pasajero está llegando. Esperando: 7
El pasajero está llegando. Esperando: 8
Llegada de 2 metrobus
El metrobús ha partido. Pasajeros: 8
El metrobús ha partido. Pasajeros: 0
Personas esperando: 0
Pulsar enter para continuar. Ingresar q para salir:
```

Figura 3: Captura de ejecución exitosa