Proyecto 2

Yoav Farid Galdamez Pozos

La situación modelada por medio del programa, es el comportamiento de un conjunto de microbuses a través de una ruta, donde en la mayoría de los casos los microbuseros compiten entre sí por el pasaje, lo que genera nuestra situación problema que es la concurrencia de varios microbuseros en una parada. Ejemplo de que los microbuses compiten para ganar el pasaje. Nuestra propuesta plantea el uso de los semáforos para poder limitar la cantidad de microbuseros en las paradas de una ruta, de tal forma que solo un microbusero pueda subir pasaje en una parada, y así poder subir y bajar más gente, y dadas las condiciones actuales de pandemia, se limita a una capacidad máxima de personas que pueden subir al microbús. Debido a que son microbuses de ruta tienen la posibilidad de rebasarse entre paradas como lo harían en la vida real. Además de agregar un checador que pueda verificar la bajada y subida segura de las personas, debido a que los microbuses son conocidos por arrancar de golpe.

¿Dónde pueden verse las consecuencias nocivas de la concurrencia?

Es muy común ver como microbuses pierden la oportunidad de llevar personas cuando más de uno llega a la parada, ocasionando que solo uno de ellos se llene o que entre los dos lleven muy pocas personas, provocando que más gente se forme esperando, probablemente, mucho tiempo a otro microbús.

¿Qué eventos pueden ocurrir que queramos controlar?:

Lo que queremos controlar con esto es que solamente un microbús llegue a una parada distinta, para que así no haya ese tipo de desperdicio de espacio y se pueda transportar más personas. Y que los conductores no compitan por el pasaje. También que se de tiempo de bajada y subida de los usuarios del microbús. También se quiere controlar el hecho de que los microbuses lleguen en orden a la base y salgan de esta sin rebasarse.

¿Hay eventos concurrentes para los cuales el ordenamiento relativo no resulta importante?:

Para esta situación no es necesario que las personas se formen o lleguen de un orden determinado para poder subir al microbús, la única condición para que puedan usarlo es que este tenga espacio, sino las personas que no hayan alcanzado a subir tendrán que esperar en la parada hasta que llegue otro microbús

.

Mecanismo de sincronización empleados:

Los mecanismos utilizados para poder resolver el problema, en primera usar clases las cuales nos ayudaran a manejar mejor los datos que se le asignen a cada hilo que simule ser un microbús. Para poder crear una ruta que tengan que seguir los microbuses y para poder contener a las personas que estarán esperando en cada parada decidimos utilizar vectores. Para poder resolver el problema de concurrencia en la base tanto para que las microbuses se formen y para que respeten turnos para subir personas, utilizamos dos mutex; uno que impide el acceso a más de un microbús a la base y otro que nos ayuda a que solo un microbús pueda empezar a subir personas. Cuando el microbús tenga un determinado tiempo esperando gente o que ya se llene su capacidad disponible tendrá que “arrancar” para empezar la ruta, teniendo así, que soltar los dos mutex para que otro microbús pueda empezar a subir gente. En el problema de cuánto tiempo deben de esperarse los microbuses en las paradas, usamos dos semáforos los cuales nos ayudan a controlar cuando una microbús está subiendo o bajando gente, mientras que para calcular el tiempo que necesita la micro para ejecutar cada acción es nuestra clase checador, la cual calcula el tiempo que tardará la micro en usar una parada, dependiendo de cuantas personas suban y bajen poniendo a dormir al hilo el tiempo calculado para que al final de la siesta suelte la parada y se encamine a la siguiente.

¿Qué eventos pueden ocurrir que queramos controlar?

Lo que se quiere controlar con esto es que solamente un microbús llegue a una parada distinta, para que así no haya ese tipo de desperdicio de espacio y se pueda transportar más personas. Y que los conductores no compitan por el pasaje. También que se de tiempo de bajada y subida de los usuarios del microbús. También se quiere controlar el hecho de que los microbuses lleguen en orden a la base y salgan de esta sin rebasarse.

¿Hay eventos concurrentes para los cuales el ordenamiento relativo no resulta importante?

Para esta situación no es necesario que las personas se formen o lleguen de un orden determinado para poder subir al microbús, la única condición para que puedan usarlo es que este tenga espacio, sino las personas que no hayan alcanzado a subir tendrán que esperar en la parada hasta que llegue otro microbús.

Mecanismo de sincronización empleados

Los mecanismos utilizados para poder resolver el problema, en primera usar clases las cuales nos ayudaran a manejar mejor los datos que se le asignen a cada hilo que simule ser un microbús. Para poder crear una ruta que tengan que seguir los microbuses y para poder contener a las personas que estarán esperando en cada parada decidimos utilizar vectores. Para poder resolver el problema de concurrencia

en la base tanto para que las microbuses se formen y para que respeten turnos para subir personas, utilizamos dos mutex; uno que impide el acceso a más de un microbús a la base y otro que nos ayuda a que solo un microbús pueda empezar a subir personas. Cuando el microbús tenga un determinado tiempo esperando gente o que ya se llene su capacidad disponible tendrá que “arrancar” para empezar la ruta, teniendo así, que soltar los dos mutex para que otro microbús pueda empezar a subir gente. En el problema de cuánto tiempo deben de esperarse los microbuses en las paradas, usamos dos semáforos los cuales nos ayudan a controlar cuando una microbús está subiendo o bajando gente, mientras que para calcular el tiempo que necesita la micro para ejecutar cada acción es nuestra clase checador, la cual calcula el tiempo que tardará la micro en usar una parada, dependiendo de cuantas personas suban y bajen poniendo a dormir al hilo el tiempo calculado para que al final de la siesta suelte la parada y se encamine a la siguiente.

Nuestros hilos microbuses constantemente están intentando entrar a una parada para poder empezar a subir o bajar gente y para terminar su recorrido, es por esto que recurrimos al uso de semáforos y candados para restringir a que estos microbuses esperen su turno para empezar a trabajar. La función que tiene el candado “mutex” es el controlar a las microbuses para que no se junten muchas en una misma parada, provocando que solo una de ellas ocupe una parada y se pueda llevar a todas las personas que esta pueda subir mientras que a su vez están bajando personas que hayan llegado a su destino, así cuando una microbús haya esperado el tiempo necesario para que las personas puedan subir y bajar sin peligro suelte el candado para dirigirse a su próxima parada dejando libre la anterior para otra microbús. Nuestros semáforos “YaBajaron” y “Ya Subieron” junto con nuestra clase “Checador” cumplen con la función de controlar la entrada y salida de las personas en la micro, para así poder determinar un tiempo en específico el cual nuestra micro tiene que esperar en la parada para que las personas puedan hacer estas acciones sin peligro alguno de que se arranque sin avisar. Descripción algorítmica del avance de cada hilo/proceso: - Checador. Es el encargado de asegurarse de que las personas puedan subirse y bajarse sin temor de que el microbús se arranque, haciendo esperarse en la parada el tiempo que sea necesario, dependiendo de cuantas personas bajen y suban. En el caso de la subida, el checador se encarga de dormir al hilo un tiempo determinado con la operación de las personas que quieran subir por un tiempo constante que tarda cada persona en subir, cuando se termina el tiempo de la siesta de la micro, indica que ya termino (release), de esta misma forma pasa con las personas que quieran bajar

```
class Checador:
    tiempo = 0
    tiempoPorPersona = 0.02

    #Método que indica cuanto tiempo se debe de esperar la micro para permitir que las personas suban
    def tiempoSubida( self, personasSuben, bus ):
        self.tiempo = personasSuben*self.tiempoPorPersona
        print( "Micro:%d Suben: %d personas"%( bus, personasSuben ) )
        sleep( self.tiempo )
        YaSubieron.release()

    #Este método indica cuanto tiempo se debe esperar la micro para que la personas se bajen
    def tiempoBajada( self, personasBajan, bus ):
        self.tiempo = personasBajan*self.tiempoPorPersona
        print( "Micro:%d Bajan: %d personas"%( bus, personasBajan ) )
        sleep( self.tiempo )
        YaBajaron.release()
```

Parada.

Esta clase nos permite ir creando paradas para los microbuses teniendo ya de inicio de creación a los checadores que estarán organizando a las microbuses para poder entrar y salir de estas

```
class Parada:
    checador = Checador()
    parada = Semaphore( 1 )
```

Bus.

Esta clase es la encargada de crear nuestros hilos microbuses, dándoles los atributos necesarios para determinar cuántas personas pueden almacenar, cuántas personas tienen en su interior y en qué parada se encuentra de la ruta e iniciando en el proceso “base”..

```
class Bus( Thread ):
    capacidad = 50
    paradas = 0

    def __init__( self, Nombre, Paradas ):
        super().__init__( target = base, args = [] )
        self.personas = 0
        self.parada = 0
        self.nombre = Nombre
        self.paradas = Paradas
```

paradaSube.

Con este proceso calculamos aleatoriamente las personas que subirán al microbús en cada parada.

```
def paradaSube( parada, capacidad ) -> int :
    global personasEnParadas
    personasEnParadas[ parada ]  = random.randint( 0, capacidad-42 )
    return personasEnParadas[ parada ]
```

paradaBaja.

Proceso encargado de determinar aleatoriamente las personas que bajarán en cada parada del microbús, en base a la cantidad de personas que están dentro de la misma.

```
def paradaBaja( personas ) -> int:
    return random.randint( 0, personas )
```

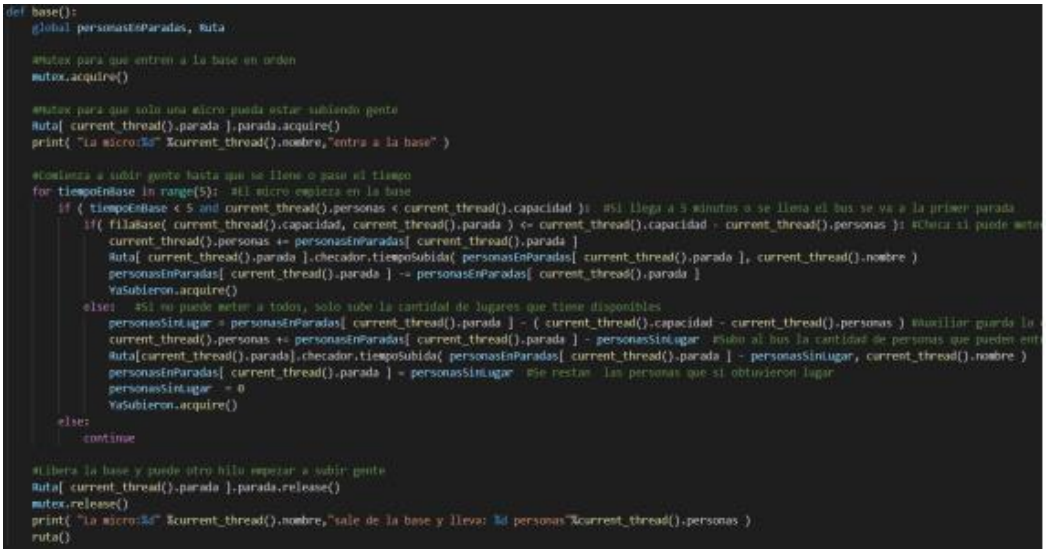
filaBase.

Proceso encargado de calcular aleatoriamente cuantas personas subirán al microbús estando en la base, por ende este proceso es ligeramente distinto a ***"paradaSube"*** ya que en la base tiende a haber más personas que en las paradas intermedias entre bases y base.

```
def filaBase( capacidad, parada ) -> int:
    global personasEnParadas
    personasEnParadas[ parada ] += random.randint( 0, capacidad-35 )
    return personasEnParadas[ parada ]
```

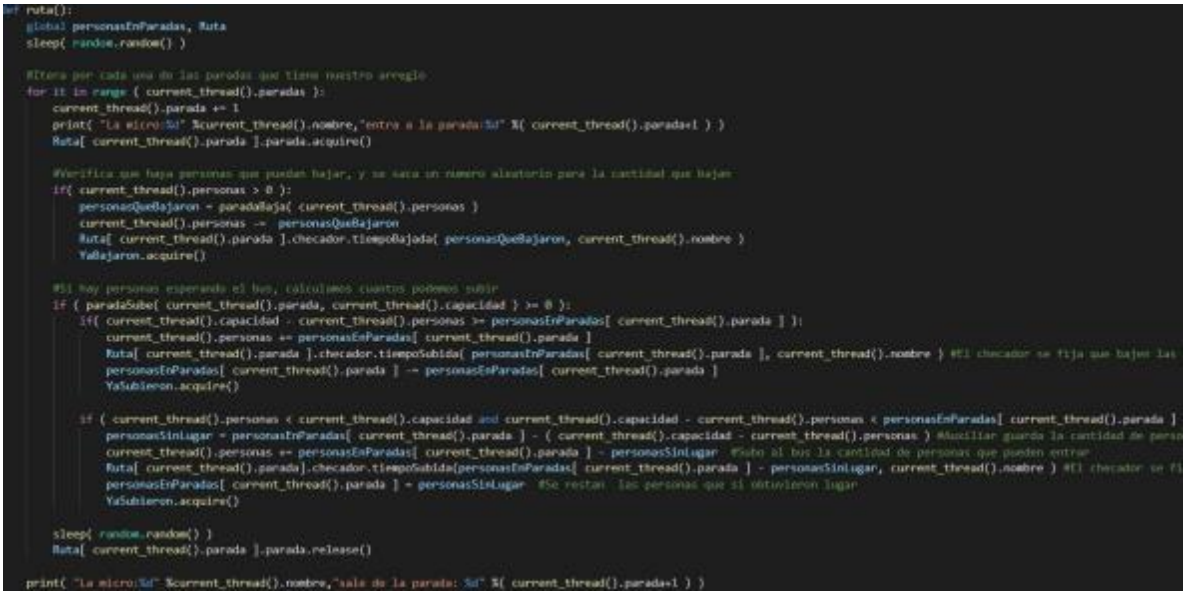
base.

Este proceso utiliza el candado "mutex" para poder organizar a los microbuses, de tal modo que sólo una pueda entrar a empezar a subir gente, de esta forma los microbuses siempre saldrán de la base de forma ordenada, liberando el candado cada vez que salga una.



ruta.

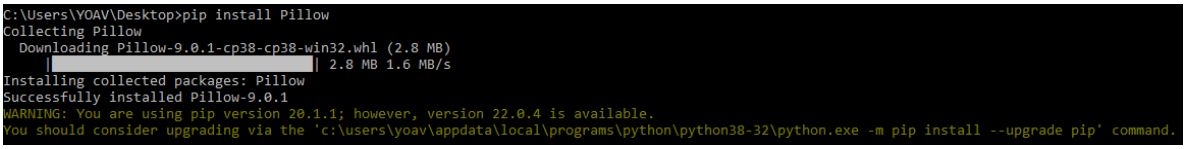
Es nuestro proceso que simula todas las paradas que deberán seguir los microbuses una vez salido de base, usando las variables globales “personasEnParada” y “Ruta” las cuales nos ayudaran a ver cuántas personas están esperando en cada parada determinada e iterando en cada una de las paradas que tenga nuestro vector, cada vez que un hilo llegue a la parada agarrará al semáforo que tiene por la clase “Parada” evitando así que otro hilo pueda entrar hasta que este termine de bajar y subir gente mediante el uso del checador y de los procesos “paradaBaja” y “paradaSube”, una vez haya pasado el tiempo que se necesito para que la gente baje y suba el hilo soltara el semáforo de esa parada para que otro pueda entrar.



Descripción del entorno de desarrollo.

¿Qué lenguaje emplean? ¿Qué versión?

El lenguaje empleado fue Python 3.9.7 Se ejecuta con el comando en el cmd “py P1Micros” ¿Qué bibliotecas más allá de las estándar del lenguaje?: Para poder utilizar la biblioteca PIL se requiere instalar Pillow poniendo el siguiente comando en el cmd “pip install Pillow”



from tkinter import * from PIL import ImageTk, Image from threading import Semaphore, Thread, current_thread from time import sleep import random ¿Bajo

qué sistema operativo / distribución lo desarrollaron y/o probaron?: Este proyecto lo desarrollamos y probamos en windows 10.

Ejemplos o pantallazos de una ejecución exitosa.

Para tener una ejecución exitosa se requiere que descargue la imagen disponible en esta carpeta con el nombre “A.jpg” y guardarla en la misma carpeta que guardó el código.