Members	Ryan Onil Barrion	Date Performed	4/17/2023
	Rayvhen Mico Rosima		
	Juan Diego Javier		

Synchronization Technique

The group used bounded semaphores. Using bounded semaphores provides a simple mechanism to limit the threads in such a way that follows the constraints given. Bounded semaphores help avoid the situation where there are too many “customers” or threads inside the fitting room. If this occurs, then many problems such as long delay times and deadlocks might arise. It enables a more efficient management of shared resources.

Variables

Global Variables

```
global maxSlot
global customersLeft
global customersInside
global blueThreads, greenThreads

global maxSlot
```

```
class ThreadID:
    def __init__(self, id, color):
        self.id = id
        self.color = color
```

```
def dressingRoom(ThreadID, limit, curr_color):
    global maxSlot
    global customersLeft
    global customersInside
    global blueThreads, greenThreads


   ….
```

The parameters for the dressingRoom include ThreadID, limit, and curr_color. ThreadID refers to the current thread being passed through the function. Limit refers to the maximum number of customers/threads there can be in the fitting room. The curr_color variable sets what color threads can enter the fitting room. Both limit and curr_color are important to deal with the constraint A.

The code highlighted in red refers to the global variables that the dressingRoom function will use. The maxSlot variable refers to the semaphore that is initialized in the main function. The customersLeft variable refers to how many customers/threads have not yet gone through the fitting room. The customersInside variable refers to how many customers/threads that are currently in the fitting room. Both the greenThreads and blueThreads variables are lists of “ThreadID” defined previously. To explain further, “ThreadID” has two variables: ID and color where ID is a number that corresponds to a specific thread and the color refers to the color of the thread itself (either blue or green).

Constraint Solution

Constraint A

“There are only n slots inside the fitting room of a department store. Thus, there can only be at most n persons inside the fitting room at a time.”

In the code, the maximum number of customers (limit) and a count of how many customers are in the fitting room (customersInside). With these two variables, if the number of customers inside the fitting room is equal to the limit of the fitting room it means that it is time to release customers from the fitting room. In addition to this, when initializing the semaphore it is also setting the limit of the fitting room.Furthermore, in the main code, a semaphore object was created,specifically the “BoundedSemaphore”, this ensures that only a max of n threads can enter the dressing room and the other threads get locked out while the semaphore is still full.

```
if ((customersInside == limit)):
        if curr_color == 'Blue':
            while(len(blueThreads) > 0):
                print(f'{blueThreads[0].color} {blueThreads[0].id} has left the dressing room')
                blueThreads.pop(0)
            if b > 0:
                print('Dressing Room Empty...')
            maxSlot.release(customersInside)

        else:
            while(len(greenThreads) > 0):
                print(f'{greenThreads[0].color} {greenThreads[0].id} has left the dressing room')
                greenThreads.pop(0)
            if g > 0:
                print('Dressing Room Empty...')
            maxSlot.release(customersInside)

        customersInside = 0
```

```
def main():
    n = int(input('Enter the number of slots in the fitting room:'))
    b = int(input('Enter the number of blue threads:'))
    g = int(input('Enter the number of green threads:'))

    global maxSlot
    maxSlot = BoundedSemaphore(n)
```

Constraint B

“There cannot be a mix of blue and green in the fitting room at the same time. Thus, there can only be at most n blue threads or at most n green threads inside the fitting room at a time.”

When creating and running threads for each color, there will be a small time delay of 1 second. Creating a time delay, allows time for thread of a certain color to finish using the dressing room first and leave, then allow the next thread to enter the dressing room immediately once a free slot or space in the semaphore is found.

```
    # Creates a thread
    thread = Thread(target=dressingRoom, args = [blueThread, n, curr_color,b,g])
    thread.start()

    # Adds a small time delay to prevent mixing of colors inside the dressing room
    time.sleep(1)
    handleBlueThreadsList.append(thread)
```

```
            thread = Thread(target = dressingRoom, args = [greenThread, n, curr_color,b,g])
            thread.start()

            # Adds a small time delay to prevent mixing of colors inside the dressing room
            time.sleep(1)
            handleGreenThreadsList.append(thread)
```

In the dressingRoom function, there is a parameter called “curr_color” which sets the only color thread that is allowed in the fitting room. With this, only the appropriate customers/threads are entering the fitting room.

```
if ((customersInside == limit) or (customersInside == g and curr_color == 'Green') or(customersInside == b and curr_color == 'Blue')):
        if curr_color == 'Blue':
            while(len(blueThreads) > 0):
                # Prints out the threads inside the dressing room to indicate that they are leaving

                print(f'{blueThreads[0].color} {blueThreads[0].id} has left the dressing room')

                # Pops the first item in the list everytime to mantain order of arrival
                blueThreads.pop(0)

            if b > 0: # As long as there are blue customers that left on a certain time period will print out to indicate the semaphore is free again
                print('Dressing Room Empty...')
            # Release the semaphore
            maxSlot.release(customersInside)

        else:
            while(len(greenThreads) > 0):
                print(f'{greenThreads[0].color} {greenThreads[0].id} has left the dressing room')

                # Pops the first item in the list everytime to mantain order of arrival
                greenThreads.pop(0)

            if g > 0: # As long as there are blue customers that left on a certain time period will print out to indicate the semaphore is free again
                print('Dressing Room Empty...')
            # Release the semaphore
            maxSlot.release(customersInside) # Remainder Section

        # Empties out the dressing room
        customersInside = 0
```

Constraint C

“The solution should not result in deadlock.”

The solution prevents deadlock through the use of semaphores. In this case, the “maxSlot” semaphore limits the number of threads (customers) that can enter the fitting room at any given time. By using a bounded semaphore here, the number of threads is limited to the number of available slots in the fitting room which prevents multiple threads from entering simultaneously which can potentially cause a deadlock.

```
def dressingRoom(ThreadID, limit, curr_color,b,g):
    global maxSlot
    global customersLeft
    global customersInside
    global blueThreads, greenThreads


    maxSlot.acquire()

    customersLeft = customersLeft - 1

    if curr_color == 'Green':
        greenThreads.append(ThreadID)
    else:
        blueThreads.append(ThreadID)
```

```
        print(f'{blueThreads[0].color} {blueThreads[
        blueThreads.pop(0)
    if b > 0:
        print('Dressing Room Empty...')
    maxSlot.release(customersInside)

else:
    while(len(greenThreads) > 0):
        print(f'{greenThreads[0].color} {greenThread
        greenThreads.pop(0)
    if g > 0:
        print('Dressing Room Empty...')
    maxSlot.release(customersInside)

customersInside = 0
```

```
def main():
    n = int(input('Enter the number of slots in the fitting room:'))
    b = int(input('Enter the number of blue threads:'))
    g = int(input('Enter the number of green threads:'))

    global maxSlot
    maxSlot = BoundedSemaphore(n)
```

Constraint D

"The solution should not result in starvation. For example, blue threads cannot forever be blocked from entering the fitting room if green threads are lining up to enter as well."

The solution for constraint D is done by by constantly switching between threads after letting each thread color enter the dressing room at most n.

```
while customersLeft > 0:
    i = 0
    if curr_color == 'Blue':
        while i < n and b > 0:
            blueThread = ThreadID(blueCounter, curr_color)

            thread = Thread(target=dressingRoom, args = [blueThread, n, curr_color,b,g])
            thread.start()
            time.sleep(1)
            handleBlueThreadsList.append(thread)

            i = i + 1
            b = b - 1
            blueCounter = blueCounter + 1

        for t in handleBlueThreadsList:
            t.join()

        curr_color = 'Green'

    else:
        while i < n and g > 0:
            greenThread = ThreadID(greenCounter, curr_color)
            thread = Thread(target = dressingRoom, args = [greenThread, n, curr_color,b,g])
            thread.start()
            time.sleep(1)
            handleGreenThreadsList.append(thread)
            i = i + 1
            g = g - 1
            greenCounter = greenCounter + 1

        for t in handleGreenThreadsList:
            t.join()
        curr_color = 'Blue'
```