CSOPESY S13 **06/14/2022**
Group 11 **Sir Arren Antioquia**
Espiritu, Paolo Edni Andryn V.
Ponce, Andre Dominic H.

Project 2 Documentation

Synchronization Technique

In order to address the task, a combination of **mutex locks** and **semaphores** from the **threading** library of Python was used.

Synchronization Variables

Variables	Purpose
fitting_room	A **bounded semaphore** variable to simulate ***n*** slots inside the fitting room.
quantum	An **integer variable** that **limits** how many blue or green threads at a time may enter the fitting room before giving the room to the other color (if applicable).
blue_semaphore *green_semaphore*	**Semaphore** variables to accommodate the allotted slots for blue or green threads.
blue_exec_ctr *green_exec_ctr*	**Integer** variables to **count** the number of blue or green threads that finished executing.
blue_ctr_mutex *green_ctr_mutex*	**Lock** variables to securely **update** the thread counters namely, ***blue_exec_ctr*** and ***green_exec_ctr***.
blue_room_mutex *green_room_mutex*	**Lock** variables to **prevent** the opposite color, blue or green, to access the fitting room if there is currently a color, green or blue, using it.

Code & Constraints

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.

The solution handles constraint A by using a **bounded semaphore**, with an initial value of n – which is the **number of slots in the fitting room**. The difference between a bounded semaphore and a regular semaphore in the context of this library is that it does not allow the value of a bounded semaphore to exceed its initial value. This ensures that the maximum slots for the fitting room will always be n. This is shown in the code below:

```
# initialize n slots inside fitting room
fitting_room = threading.BoundedSemaphore(value=n)
```

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.

As for constraint B, the solution handles it by utilizing **two semaphores** (***blue_semaphore*** and ***green_semaphore***) and **two mutex locks** (***blue_room_mutex*** and ***green_room_mutex***). These, along with the fitting room bounded semaphore, limit the flow of threads into the fitting room in two phases. First, the color-specific semaphore controls which same-colored threads get to enter the fitting room next. Afterwards, once the color-specific mutex lock is released, those threads that were able to acquire a lock from the semaphore will finally be able to enter the fitting room. This two-step phase ensures that no blue and green threads will meet inside the fitting room. This was also made possible using the initialization for the semaphores shown below wherein one color-specific semaphore gets an initial value of ***quantum***, while the other gets **0**:

```
# limit value per "turn"
quantum = n

# if there are no green threads,
if b > 0 and g == 0:
    # just execute the blue threads
    blue_semaphore = threading.Semaphore(value=quantum)
    green_semaphore = threading.Semaphore(value=0)
# if there are no blue threads,
elif g > 0 and b == 0:
    # just execute the green threads
```

```
    blue_semaphore = threading.Semaphore(value=0)
    green_semaphore = threading.Semaphore(value=quantum)
# if there are more green threads than blue threads,
elif b < g:
    # then blue threads goes first
    blue_semaphore = threading.Semaphore(value=quantum)
    green_semaphore = threading.Semaphore(value=0)
# if there are more blue threads than green threads,
elif g < b:
    # then green threads goes first
    blue_semaphore = threading.Semaphore(value=0)
    green_semaphore = threading.Semaphore(value=quantum)
# if the number of blue threads are equal to the number of green threads,
elif b == g:
    # then default to blue
    blue_semaphore = threading.Semaphore(value=quantum)
    green_semaphore = threading.Semaphore(value=0)
# else,
else:
    print("Invalid input! There must be at least one green or blue thread!")

...

# fitting room color access mutex lock
blue_room_mutex = threading.Lock()
green_room_mutex = threading.Lock()
```

C. The solution should **not result** in a **deadlock**.

As mentioned earlier, the initialization of the ***blue_semaphore*** and ***green_semaphore*** is either the value of the ***quantum*** or **0** which depends on the number of green threads and blue threads. Now, the code segment highlighted below prevents deadlocks since the solution releases ***quantum*** number of threads to be acquired by the semaphore of the next color that will be accessing the room. This ensures that the color-specific semaphore will not run out of resources upon executing the threads it contains. Aside from that, the blue threads do not require resources from green threads so that it could be executed and vice versa. Color-specific threads only need to signal a number of threads of the opposite color to access the fitting room.

```
# TL;DR make the green threads give the blue threads a chance to execute
```

```
# if all green threads are done executing already OR
# quantum has been reached AND there are still blue threads waiting
if (green_exec_ctr % quantum == 0 and blue_exec_ctr <= b)
   or green_exec_ctr == g:

   # if there are blue threads waiting,
   if blue_exec_ctr < b:
      # release semaphores for blue threads
      for i in range(quantum):
         blue_semaphore.release()

         # release green mutex lock
         green_room_mutex.release()

   # signal that the room is empty
   print("Empty fitting room.")
```

```
# TL;DR make the blue threads give the green threads a chance to execute
# if all blue threads are done executing already OR
# quantum has been reached AND there are still green threads waiting
if (blue_exec_ctr % quantum == 0 and green_exec_ctr <= g)
   or blue_exec_ctr == b:

   # if there are green threads waiting,
   if green_exec_ctr < g:
      # release semaphores for green threads
      for i in range(quantum):
         green_semaphore.release()

         # release blue mutex lock
         blue_room_mutex.release()

   # signal that the room is empty
   print("Empty fitting room.")
```

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

The code segment highlighted below handles starvation since it only executes ***quantum*** number of blue or green threads at a time. It is handled by implementing the condition ***green_exec_ctr % quantum == 0*** to limit the execution of green threads by only running ***quantum*** green threads at a time before passing the fitting room access to the blue threads. Whereas the condition, ***blue_exec_ctr % quantum == 0*** only executes ***quantum*** blue threads at a time before passing the fitting room access to the green threads. It also checks if there are opposite-colored threads to be executed so that the program would know if it is necessary to transfer the access to the fitting room. Moreover, it verifies if the number of executed color-specific threads already reached its max value so that it would only process the opposite-colored threads until the end of the program.

```
# TL;DR make the green threads give the blue threads a chance to execute
# if all green threads are done executing already OR
# quantum has been reached AND there are still blue threads waiting
if (green_exec_ctr % quantum == 0 and blue_exec_ctr <= b)
   or green_exec_ctr == g:

   # if there are blue threads waiting,
   if blue_exec_ctr < b:
      # release semaphores for blue threads
      for i in range(quantum):
          blue_semaphore.release()

          # release green mutex lock
          green_room_mutex.release()

   # signal that the room is empty
   print("Empty fitting room.")
```

```
# TL;DR make the blue threads give the green threads a chance to execute
# if all blue threads are done executing already OR
# quantum has been reached AND there are still green threads waiting
if (blue_exec_ctr % quantum == 0 and green_exec_ctr <= g)
   or blue_exec_ctr == b:

   # if there are green threads waiting,
```

```
    if green_exec_ctr < g:
        # release semaphores for green threads
        for i in range(quantum):
            green_semaphore.release()

            # release blue mutex lock
            blue_room_mutex.release()

    # signal that the room is empty
    print("Empty fitting room.")
```