CS347 Quiz 3 (take home) submission
Abhijeet Prasad Bodas
190100004

Problem statement (Q25 from practice problems)

Consider a scenario where a bus picks up waiting passengers from a bus stop periodically. The bus has a capacity of `K`. The bus arrives at the bus stop, allows up to `K` waiting passengers (fewer if less than `K` are waiting) to board, and then departs. Passengers have to wait for the bus to arrive and then board it. Passengers who arrive at the bus stop after the bus has arrived should not be allowed to board, and should wait for the next time the bus arrives. The bus and passengers are represented by threads in a program. The passenger thread should call the function `board()` after the passenger has boarded and the bus should invoke `depart()` when it has boarded the desired number of passengers and is ready to depart.

The threads share the following variables, none of which are implicitly updated by functions like `board()` or `depart()`.

```
mutex = semaphore initialized to 1.
bus_arrived = semaphore initialized to 0.
passenger_boarded = semaphore initialized to 0.
waiting_count = integer initialized to 0.
```

Below is given synchronized code for the passenger thread. You should not modify this in any way.

```
down(mutex)
waiting_count++
up(mutex)
down(bus_arrived)
board()
up(passenger_boarded)
```

Write down the corresponding synchronized code for the bus thread that achieves the correct behavior specified above. The bus should board the correct number of passengers, based on its capacity and the number of those waiting. The bus should correctly board these passengers by calling up/down on the semaphores suitably. The bus code should also update waiting count as required. Once boarding completes, the bus thread should call `depart()`. You can use any extra local variables in the code of the bus thread, like integers, loop indices and so on. However, you must not use any other extra synchronization primitives.

Correct code using semaphores

```
down(mutex)
int num_passengers = 0
while (waiting_count ≥ 0 && num_passengers ≤ K) {
    num_passengers++
    waiting_count--
    up(bus_arrived)
```

```
    down(passenger_boarded)
}
depart()
up(mutex)
```

Interleavings

(Red = passenger code, Green = bus code)

Passengers coming after bus arrived are rejected

```
// ...
    down(mutex) // mutex = -1
down(mutex) // passengers coming after the bus are kept waiting
.
.
.
    up(mutex)   // bus leaves
waiting_count++
up(mutex)       // This will wake up another passenger which was kept waiting
                // because he came after bus had arrived
// ...
```

Only upto K passengers allowed to board once

```
// ...
down(bus_arrived) // Assume this is the K+1 passenger who is waiting, and now bus
                  // arrives
    down(mutex)  // Stop new passengers, `waiting_count` cannot increase now
    int num_passengers = 0
    while (waiting_count ≥ 0 && num_passengers ≤ K) {
        num_passengers++
        waiting_count--
        up(bus_arrived) // Wake up one waiting passenger
        down(passenger_boarded) // Wait for him to board() the bus
    }
    // As can be seen, the bus wakes up only upto `K` waiting
    // passengers because of the loop condition
// ...
```

Incorrect code using semaphores (1)

```
int num_passengers = 0
while (waiting_count ≥ 0 && num_passengers ≤ K) {
    num_passengers++

    down(mutex)
    waiting_count--
    up(mutex)

    up(bus_arrived)
    down(passenger_boarded)
}
depart()
```

Consider the follow interleaving:

```
    // Bus arrives
    int num_passengers = 0
    while (waiting_count ≥ 0 && num_passengers ≤ K) {
        num_passengers++
down(mutex)
waiting_count++
up(mutex)
    down(mutex)
    waiting_count--
    up(mutex)
    // In this scenario, the bus will end up taking in even those
    // passengers which came after the bus arrived, because the bus
    // does not acquire the mutex lock just after it arrives
//  ...
```

Incorrect code using semaphores (2)

```
down(mutex)
int num_passengers = 0
while (waiting_count ≥ 0 && num_passengers ≤ K) {
    num_passengers++
    waiting_count--
    down(passenger_boarded) // The order of these
    up(bus_arrived)         // two lines has been interchanges
}
depart()
up(mutex)
```

The above code will cause a **deadlock**, because the bus will wait for the passenger to `board()`, and the passenger will wait for the bus to arrive.

Correct code using condition variables

Assuming the same variables as earlier. The given passenger code can be written using condition variables as:

```
lock(mutex)
waiting_count++
unlock(mutex)
wait(bus_arrived)   // There is no condition to check before waiting
board()
signal(passenger_boarded)
```

The correct bus code will be:

```
lock(mutex)
int num_passengers = 0
while (waiting_count ≥ 0 && num_passengers ≤ K) {
    num_passengers++
    waiting_count--
    signal(bus_arrived)
    wait(passenger_boarded)
}
depart()
lock(mutex)
```