Sincronización

Ejercicio 3

Nos han pedido que hagamos un simulador de vuelo.
Más precisamente para un controlador áereo. Este controlador es el encargado de hacer aterrizar aviones en pistas.
El simulador funciona de la siguiente manera:
Se tienen N aviones, que serán representados por procesos y una pista de aterrizaje. Nunca una pista puede estar siendo utilizada por dos aviones al mismo tiempo, porque ya sabemos qué pasaría.
Los aviones comenzarán en vuelo. Puede asumir que cada proceso de avión toma como parámetro el índice `i` correspondiente a su id de avión y que el mismo irá de 0 a $N-1$.
Para aterrizar, los aviones llamarán a la función `pedir_pista()`. Los aviones podrán aterrizar llamando a la operación `aterrizar()`. Tenga en cuenta que sólo podrán hacerlo si la pista está vacía. De no estarlo deberán esperar en orden del llamado `pedir_pista()`.
Al terminar de aterrizar deberán llamar a `aterrizé_bien()` y luego podrá aterrizar el siguiente avión en el orden descripto.
Una vez hayan aterrizado todos los aviones, se procede al famoso "asadazo de camaradería" entre todos los pilotos llamando (curiosamente desde el avión) a la función `asadear()` para luego poder despegar nuevamente con la función `despegar()` y volver a comenzar. No deberá bloquearse más de lo necesario.
Sólo puede utilizar mutexes, semáforos y tipos de datos básicos atómicos. Cualquier otra estructura que desee utilizar deberá definirla y escribir su código primero. Puede utilizar el tipo cola con todas las operaciones típicas (encolar, desencolar, buscar_elemento, iesimo, etc)

- Realizar el pseudocódigo del simulador.
- Extender el pseudocódigo realizado anteriormente para poder tener M pistas

Solución

```
mutex = sem(1)
mutex2 = sem(1)
barrera = sem(0)
sem_aterriza = [sem(0), ..., sem(0)]
int llegando = llegaron = 0
queue<int> cola

avion(i):
  mutex.wait()
    llegando++
    pedir_pista()
    if(llegando == 1){
      sem_aterriza[i].signal()
    } else {
      cola.encolar(i)
    }
  mutex.signal()

  sem_aterriza[i].wait()
  aterriza()
  aterrize_bien()

  mutex.wait()
    llegando--
    if(cola.size() != 0) {
      int siguiente = cola.top()
      cola.desencolar()
      sem_aterriza[siguiente].signal()
    }
  mutex.signal()

  mutex2.wait()
  llegaron++
  if(llegaron == N){
    barrera.signal(N)
  }
  mutex2.signal()

  barrera.wait()
  asado()
  despegar()
```

El ejercicio tiene un par de ideas centrales. Como nos importa bastante el orden en que fueron llamando a *pedir_pista*, entonces una cola es una estructura apropiada para este problema para saber el orden en que se fueron apilando los llamados. Podríamos pensar que usando simplemente un semaforo, el problema podria ser resuelto. Sin embargo, no podemos asumir nada acerca de cómo funcionan, no sabemos en qué orden va a ir despertando a los procesos. Cuando un avión aterriza bien, entonces deberá desencolar un elemento de la cola, para saber quien es el siguiente. Para despertar a ese en específico, tenemos un arreglo de semáforos. La i-ésima posición indica el semáforo del i-ésimo avión. De esta forma, cuando tengamos la información de quien va siguiente, simplemente podemos despertar al procesos correspondiente en su posición dada.

Hay un detalle con respecto a como hacemos con el primero que llegue, que no debería estar en colado. Por eso tenemos una variable llegando, que va contando cuántos aviones están cerca de la pista de aterrizaje. Si solo llego uno, entonces ese debe darse un *signal* a sí mismo (para dejarse pasar) y evitar encolarse. Hay que tener cuidado con luego disminuir esa variable en el momento adecuado. Para revisar que no estemos haciendo top de una cola vacía, debemos comprobar su tamaño. Como observación, es importante notar que el mutex tanto cuando se encola, como se desencole, tiene que ser él mismo, sino podríamos tener problemas de concurrencia.

Por último, se implementa una barrera como la vista en clase para el tema del asado.