UNIVERSIDAD DIEGO PORTALES

23 de Noviembre, 2019

Tarea 2 - Sincronización

Alumno : Ulises Peña

Profesor : Martín Gutiérrez

Ayudante : Yerko Ortiz

I. Solución 1 (Herramientas de sincronización)

I.I. Uso

- Compilación: `g++ -o main1 main1.cpp -lpthread`
- Ejecución: `./main1 m n k j step`, donde:
  - `m`: Ancho del mapa.
  - `n`: Alto del mapa.
  - `k`: Número de ciudadanos *moderados*
  - *j*: Número de ciudadanos *flaites*
  - `step`: Tiempo (en milisegundos) entre cada iteración de los threads.

I.II. Descripción de la solución

Primeramente, se define una clase Casilla que contiene información acerca de si está saqueada o no, qué ciudadano y qué prófugo se ubica en ella.
Luego, se define al mapa (Chale), como un vector global bidimensional que contendrá casillas, que más tarde es inicializado con dimensiones `m` y `n`. Se utiliza un vector y no un arreglo, ya que se requiere que el mapa sea global, pero con las dimensiones entregadas por los parámetros de ejecución, que solamente están disponibles en la función `main()` mediante `argv[]`.

A continuación, sabiendo que tendremos `k+j+2` personajes moviéndose por las casillas, determinamos aleatoriamente sus posiciones de *spawn* en el mapa. Se define un vector de enteros de tamaño `m*n` con valores de 0 hasta `m*n-1`, los cuales representan el índice de las casillas en el mapa. Cada índice representa la casilla `Chale[index/m][index%m]`. De esta manera, para cada personaje escogemos un índice aleatorio del vector mediante `v[rand() % v.size()]`, luego removimos el índice escogido del vector.

Ya con todos los personajes ubicados en el mapa, estos pueden moverse según las condiciones especificadas (moderados y prófugos no pueden acceder a casillas saqueadas, 1 ciudadano y 1 prófugo por casilla, etc).
Para sincronizar a los hilos al moverse por el mapa, se implementan diferentes *mutexes*. Se definen dos arreglos de *mutexes* para cada casilla del mapa, un arreglo para los ciudadanos y el otro para los prófugos. Al *spawnear* cada personaje en su casilla,

toma control del *mutex* de la casilla, al intentar moverse a otra casilla (en una vecindad de Von Neumann), intenta tomar control del *mutex* de aquella casilla mediante `locks[y][x].try_lock()`, función que retornará `true` si se pudo tomar control del *mutex*, o `false` si no. Si se toma control el *mutex*, entonces el personaje puede moverse tranquilamente. Una vez que se trasladó de casilla, el personaje libera el *mutex* de su posición anterior (Dependiendo de si el personaje es ciudadano o prófugo, estos consultan los *mutexes* que corresponden a su tipo, de forma que entre clases no se toparán, pero sí entre ciudadanos y prófugos).

Para verificar si los prófugos han sido capturados, estos intentan tomar el control del *mutex* de la casilla en la que están, pero de los ciudadanos. Si no se puede adquirir el *mutex* significa que en la misma casilla se ubica un ciudadano por lo que ha sido capturado. Se obtiene el tipo del ciudadano (moderado o flaite), y se realiza lo correspondiente para cada situación.

Se lleva registro de la captura de los prófugos con *booleanos* para cada prófugo. Cuando todos estos son verdaderos, se activa un *booleano* para terminar la ejecución de los hilos, terminando así el programa.

II. Solución II (Monitor)

II.I. Uso

- Compilación: `g++ -o main2 main2.cpp -lpthread`
- Ejecución: `./main2 m n k j step`

II.II. Descripción de la solución

Esta solución comparte las mismas clases que la solución 1, y las posiciones de `spawn` de los personajes se determina de la misma manera.
Para esta solución se implementa una clase *Monitor*, el cual está encargado de supervisar a los hilos, de forma que no ocurra ninguna situación no deseada. Para la implementación de este, se utiliza una *variable de condición* junto a un *mutex*. En cada iteración, los hilos esperan en la variable de condición (`cv.wait(lock)`) y el monitor los llamará uno a uno (`cv.notify_one()`), el propósito del *mutex* es que los hilos llamen para esperar en la variable de condición atómicamente.

Cuando es el turno de moverse de un personaje, este llama a una función del monitor pidiéndole moverse a cierta posición deseada. El monitor comprobará si el movimiento es posible y válido, y moverá (o no) al personaje y le retornará si se realizó el movimiento, además de información a los prófugos si estos han sido capturados.

Para terminar la ejecución de los hilos, se implementan los mismos *booleanos* que en la solución 1.

Fig. 1: Programa en ejecución y finalizado